University of Alberta

K-Nearest Neighbours with weighted linear Regression

by

J. M. David Bullas

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 1998

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-34340-5

Canada

University of Alberta

Library Release Form

Name of Author: J. M. David Bullas

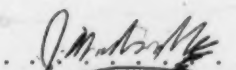
Title of Thesis: K-Nearest Neighbours with weighted linear Regression

Degree: Master of Science

Year this Degree Granted: 1998

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



J. M. David Bullas
#110, 4605 - 106A Street
Edmonton, Alberta
Canada, T6H 5H1

Date: Oct. 1/98

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **K-Nearest Neighbours with weighted linear Regression** submitted by J. M. David Bullas in partial fulfillment of the requirements for the degree of **Master of Science**

Dr William Armstrong

Dr Jim Hoover

Dr Edward Lozowski

Date: Sept. 14, 1998

I met a traveller from an antique land,
Who said - "Two vast and trunkless legs of stone
Stand in the desert . . . Near them, on the sand,
Half sunk, a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them, and the heart that fed;
And on the pedestal, these words appear:
'My name is Ozymandias, King of Kings,
Look on my Works, ye Mighty, and despair!'
Nothing beside remains. Round the decay
Of that colossal Wreck, boundless and bare
The lone and level sands stretch far away.
Percy Bysshe Shelley (1792-1822)

Abstract

The K-Nearest Neighbours method was one of the first attempts to get computers to do pattern recognition. However, the computational costs were at the time very high and computer scientists looked for alternatives to speed up the computations.

In this thesis, we show that the K-Nearest Neighours approach is still a powerful tool. By using some of the enhancements that have been developed by other scientists and by adding a twist of our own, we take the KNN and transform it from a classification tool into a tool that can perform regression. We show that it performs at least as well as KNN in its original use as a pattern classification tool, and then we go on to explore its effectiveness doing regression.

Once we are satisfied that K-Nearest Neighbors with weighted linear Regression is effective in learning regression problems, we attempt to use it in conjunction with ALNs to meld the accuracy of KNNR with the speed of the ALNs. We then use KNNR on a difficult real world data set and see how well it performs.

To my wife, may she find love, honesty and happiness for all her days for the patience and understanding she has shown.

Acknowledgements

Many people have helped me get this thesis finished. I'm sure to miss people whose names I should include, but I'm going to give it a try anyways. First, I'd like to thank my wife for uncountable things, from her patience and understanding to her ability to get me to stop thinking about my thesis. I'd like to thank my thesis advisor, Dr. Bill Armstrong, for his time and effort in wading through my torturous prose. Lastly, I'd like to thank my family and friends, without whose presence, support and companionship I would have lost my sanity and, in all likelihood, ended up as a hermit.

Contents

1 Introduction 1

2 Pattern Recognition Background 3
- 2.1 Common Pattern Recognition Issues . . . 3
  - 2.1.1 Classification and Regression . . . 3
  - 2.1.2 Speed vs Accuracy . . . 4
  - 2.1.3 The Curse of Dimensionality . . . 5
  - 2.1.4 Sampling . . . 6
  - 2.1.5 Error Metrics . . . 8
  - 2.1.6 Bayes Decision Theory . . . 9
  - 2.1.7 Decision Boundaries and Discriminant Functions . . . 11
- 2.2 Methods . . . 12
  - 2.2.1 Linear Discrimination . . . 12
  - 2.2.2 Linear Regression . . . 14
  - 2.2.3 Neural Networks . . . 16
  - 2.2.4 Adaptive Logic Networks . . . 19
- 2.3 K-Nearest Neighbours . . . 21
  - 2.3.1 The idea and the algorithm . . . 21
  - 2.3.2 Early Enhancements . . . 22
  - 2.3.3 Fast KNN . . . 23
  - 2.3.4 Weighted KNN . . . 23
  - 2.3.5 Modern KNN usage . . . 23

3 The KNNR Method 25
- 3.1 Purpose . . . 25
- 3.2 Step 1: Normalization . . . 25
  - 3.2.1 Human Intervention when Evaluating Automated Systems . . . 26
- 3.3 Step 2: Find the Nearest Neighbours . . . 27
- 3.4 Using the Nearest Neighbours . . . 29
- 3.5 The Weighting Function . . . 30
- 3.6 Smooth Output Values . . . 30
- 3.7 Weighted Linear Regression . . . 31
- 3.8 Determining K . . . 32

4 KNNR Tests on Synthetic Data 35
4.1 Tests Varying the Numbers of Neighbours . . . 36
4.1.1 Hypothesis . . . 36
4.1.2 Hyperplanes . . . 36
4.1.3 Trigonometric Functions . . . 37
4.1.4 Results and Conclusion . . . 37
4.2 Error Tests . . . 42
4.2.1 Hypothesis . . . 43
4.2.2 Results and Conclusion . . . 43
4.3 Training Size Tests . . . 44
4.3.1 Hypothesis . . . 44
4.3.2 Test Description . . . 45
4.3.3 Results and Conclusion . . . 45

5 Statlog 47
5.1 Introduction . . . 47
5.2 Methodology . . . 47
5.2.1 Australian dataset . . . 48
5.2.2 Diabetes dataset . . . 49
5.2.3 Satimage dataset . . . 49
5.2.4 Segment dataset . . . 49
5.2.5 Vehicle dataset . . . 49
5.3 Results . . . 50

6 Resampling using KNNR 53
6.1 The Resampling Method . . . 53
6.2 Using the ALN . . . 54
6.2.1 The ALN Training Method . . . 54
6.2.2 Our use of the ALN . . . 55
6.2.3 The Combined ALN/KNNR Method . . . 55
6.3 Results and Conclusions . . . 56

7 Ship Icing 58
7.1 Previous Work . . . 59
7.2 Our Work . . . 59
7.3 Results and Conclusions . . . 60

8 Conclusion and Future Work 61

Bibliography 77

List of Figures

2.1 Initial Data 7
2.2 Curved Data 7
2.3 Linear Data 7
2.4 Data set 12
2.5 Linear Divider 12
2.6 No Good Fit 13
2.7 No Good Divider 13
2.8 Data set 14
2.9 Regression 14
2.10 No Good Fit 14
2.11 Neural Network Topology 18
2.12 ALN Topology 20
2.13 ALN Example 20

3.1 Bucket Pruning 28
3.2 Weighting Function 28
3.3 Linear KNN 31
3.4 Linear Output 31
3.5 KNN Output 31
3.6 Overtraining 33
3.7 Better Generalization 33

4.1 2x+4 36
4.2 tanh(x) 36
4.3 1+sin(Ax) 36
4.4 Error for 1+sin(x) Neighbour Tests 38
4.5 Error for 1+sin(5x) Neighbour Tests 38
4.6 Error for 1+sin(10x) Neighbour Tests 39
4.7 Error for tanh(x) Neighbour Tests 39
4.8 Error for Linear Neighbour Tests 40
4.9 Raw 1+sin(10x) Data 41
4.10 Line with Error in Inputs 41
4.11 Hyperplanar Error Test Results 43
4.12 Error for Linear Sampling Tests 46
4.13 Error for Linear Sampling Tests 46

6.1 Resampling 1+sin(5x) 57

Chapter 1

Introduction

People learn things every day of their lives. For most of us, the learning process occurs without conscious effort and yet, for something so simple, we have a great deal of trouble understanding how this process actually works.

A number of years ago the question arose as to whether or not computers could be "taught" to learn like humans do. The idea was (and is) that if we could get computers to learn it would shed light on how humans learn and simultaneously broaden the usefulness of the computer. In the past few decades many attempts have been made to get computers to learn with varying degrees of success.

One of the key parts of the human learning process is the recognition of patterns. Humans have an incredibly powerful ability to identify patterns in the world around them and make use of them to predict further behavior, often only by observing a handful of examples. People use pattern recognition to recognize faces, repair motorcycles, spot approaching rainstorms, and identify makes and models of cars. A lot of effort has been made to try to get computers to do the same thing, that is, learn the patterns in the data presented to them and use them to predict future events or determine a classification or output.

Pattern Recognition, whether human or computer, is the process of discovering relationships in a set of data and making use of those relationships to predict missing or unknown quantities. A simple example should illustrate some of the key ideas.

Bankers regularly have to decide who to give loans to and who not to give loans to. The decision is a complicated one based on a number of factors. They use information such as how much money the applicant has in the bank, how many payments the

applicant has missed or been late on, how big the loan is, what the loan is going to be used for, how much collateral the applicant has, whether the applicant is married or not, how old the applicant is, how much income the person makes per year or per month, and many other such factors. Some factors are very important, other less so. To take all of these factors into account and to exhaustively list what to do in each possible case is impossible. Instead, the banker has some guidelines to work from. People who miss payments are less likely to get a loan than people who make all of their payments on time. People with little money in the bank are less likely to get a loan than people who have more money, and so on. The banker also looks at certain combinations, like the total income of a married couple minus their monthly expenses. Finally, the banker has his practical experience to draw on. He knows that in certain cases a loan was granted and it was a good decision, in others it was a bad decision, and in others a loan application was rejected. By finding the example or examples that are similar to the current situation and recognizing the pattern those examples contain, he draws upon his past experience to finally decide to give the loan or not.

There are many ways to look at the different factors involved in deciding whether to grant a loan. If there was only one way to combine the information to get the answer we wouldn't need to use pattern recognition to solve it, but there are many different ways of using the information presented to us. The hard part, and the part that makes computer pattern recognition a difficult and interesting area of study, is to find the most effective way of using that information to draw conclusions.

Chapter 2

Pattern Recognition Background

Pattern Recognition is a large field. So large, in fact, that it is beyond the scope of this thesis to provide an exhaustive list of all of the different tools and algorithms that do Pattern Recognition. Instead, we will present some basic background material on Pattern Recognition and then talk about some of the algorithms and their strengths and weaknesses. Before we talk about particular methods of pattern recognition we should discuss some issues that are common to all pattern recognition methods.

2.1 Common Pattern Recognition Issues

2.1.1 Classification and Regression

Pattern Recognition problems come in all shapes and sizes, but almost all of them are looking for some sort of answer. Whether the answer is a class, like the grade of a certain piece of beef, or a decimal number, like the magnitude of an earthquake, determines the type of pattern recognition tools that can be applied. Pattern Recognition techniques can be used for both of these problems, but they tend to break down into two groups.

The first group of pattern recognition techniques are classification problems. Classification problems are those for which the output is one of a small range of possibilities, like the bank loan question (yes and no being the classes) or the beef quality question (which could have three classes). Classes are often numbered 0 through n-1 and are usually independent - which means that class 0 is no closer to class 1 than class 7. Classification is often the simpler of the two problems, since the output range is discrete and thus computationally more tractable.

The second group of pattern recognition techniques are called regression problems. A regression problem is a problem for which a real-valued or continuous value is expected. Problems such as temperature prediction, speed estimation, and electrical power load prediction are all regression questions. Regression techniques are often more difficult to analyze than classification techniques, as the error functions are generally more complex. We will look at both types of techniques in Section 2.2.

These two groups of methods - classification methods and regression methods - do have some overlap. However, pattern recognition tools often concentrate on one type of problem or the other. A tool that is good at dividing a data set into one of four classes or sets may have very different methods and ideas than a tool that outputs a temperature or a probability that has continuous values.

2.1.2 Speed vs Accuracy

Different pattern recognition techniques have different goals. The trade-off that has been given the most attention in the pattern recognition literature is between speed and accuracy.

Early on in the theory of pattern recognition it was realized that by sacrificing the accuracy of the technique one could get a much faster tool. This speed up is usually accomplished by sacrificing the amount of information stored by the system, since the less information you have the faster it is to process. There is emphasis on both sides of the issue, but a lot of the work has focused on the idea that faster is better, and that sacrificing some accuracy is preferable if it results in greatly increased speed.

However, this author believes that the speed of the predictor is of much less interest than accuracy. With the constant increase in speed of modern computers, the speed of an algorithm is becoming less and less of an issue. We do not wish to say that speed is unimportant. However, it is important to note that we see this issue of speed vs accuracy from the viewpoint that accuracy is the most important metric of a pattern recognition technique's success. Only when two techniques give the same or nearly the same accuracy is speed of importance, so long as the technique can produce results in reasonable time. Certainly we are not ignoring speed completely - clearly an algorithm that runs in time exponential to the size of the input space is simply too expensive for any reasonable size of input - but for the purpose of this

thesis, an algorithm shall be said to run in reasonable time if the time is polynomial in the size of the input space. As long as the algorithms run in reasonable time we shall concentrate on their predictive accuracy.

2.1.3 The Curse of Dimensionality

More is not always better. This is particularly true in the field of pattern recognition.

When we talk about the input to a pattern recognition system there are two important size characteristics. The first is the number of data points available to the system (let's call this N), and the second is the number of characteristics that each data point possesses (D). The two together determine how much information is known about the current problem - a matter of multiplying N data points times D characteristics. One could easily make the assumption that increasing either N or D should increase prediction accuracy - the more we know about the system, the better we can do. This is unfortunately not the case.

Certainly having more data points is almost always better. We usually pay a small price in processing time in exchange for an increase in information, but the increase in processing time is usually not significant. In some cases, adding highly erroneous data points can cause the system performance to degrade as widely inaccurate information is handled, but generally the more data points the better.

The problem comes when we increase the number of characteristics that a single data point has. At first glance, this should also lead to an increase in the amount of information that we have. Let say we are trying to determine the breed of dog from some observations we have made. We would naturally try to observe not only the color of the coat but the size of the dog, its age, its eye color, the length of the claws, the length of the tail, the shape of the skull, etc. And it seems true that the more observations we make the more likely we'll find a match.

For a human, this is probably true. For a computer system this is not always the case. The reason is that as you increase the number of characteristics, you vastly increase the size of the input space. In effect, for the dog classification example the more dimensions that the input has, the more possible kinds of dog there are.

If all we know is the size of the dog, we can only break the dog into a very few categories and the accuracy of the system is going to be very good. If we add color

to the characteristics, the possible number of dogs jumps dramatically. Lets look at an extremely simplified system with binary values. Lets say the first variable tells us if the dog is bigger than a terrier or smaller. We are limited to two possible types of dogs, big dogs and small ones. Now lets add a second binary indicator, say, whether the eyes are blue or not. Suddenly we can have 4 types of dogs - big ones with blue eyes, big ones without blue eyes, small ones with blue eyes and small ones without blue eyes. If we add a third indicator value, we're up 2^3 possible types of dog. If we continue adding features, we're faced with 2^D possible values, where D is the number of binary features added. And this is for binary indicator values. For B-ary indicators we have B^D possible dogs.

Why is this so bad? Because each time the dimension increases by one we double or multiply by B the number of samples we need to effectively describe the input space. The means that in order to maintain our current levels of accuracy we need to drastically increase the number of data points we have each time we increase the number of characteristics by one. This is the so-called curse of dimensionality - that as the dimension of the input space increases, the number of data points needed to describe the space adequately increases exponentially.

So why is the human so good at classifying dogs? Humans are good because they know, from experience, which questions are important and lead to answers quicker. They can discard those indicators that are irrelevant very quickly, leaving a small handful of questions that need to be asked. In essence, they know which data to ignore. There are many places, however, where we don't have any idea which areas or questions are important, and these are the areas that pattern recognition techniques are most commonly applied. It is a difficult task for computer learning algorithms to determine which inputs are important and which are not. For these types of problems, the more characteristics the computer has to interpret, the harder it will be to get the correct answers.

2.1.4 Sampling

Sampling is the process of accumulating data. In the pattern recognition field, we make the assumption that data is accumulated fairly and with as little error as possible. Even with this assumption we have to be aware of the problems that sampling

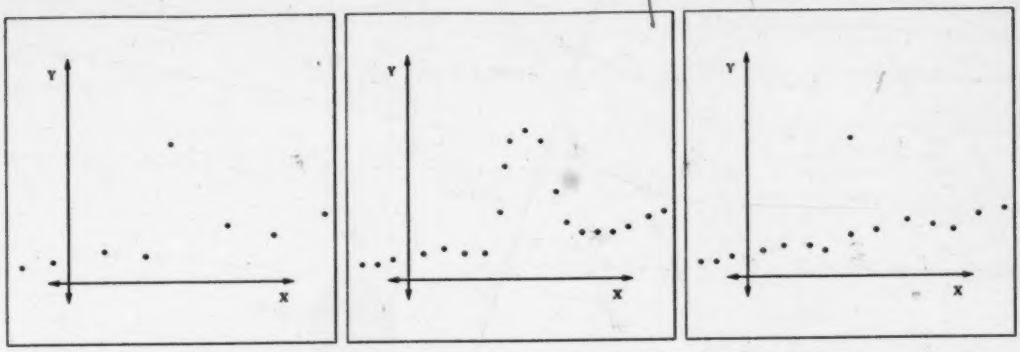


Figure 2.1: Initial Data Figure 2.2: Curved Data Figure 2.3: Linear Data

data can have.

The first problem is that occasionally, given the best of intentions, we still end up with a skewed data set. It can happen for a number of reasons from mechanical malfunction to human error to sabotage. And even if everything goes right statistics tell us that there is always a chance that we grab a misrepresentative sample from the population we are trying to study.

Further, there is the problem that sampling doesn't always tell us the whole story. Take a look at Figure 2.1. It represents a sample drawn from some simple population where the x-axis represents the input value and the y-axis represents the output at that value and there is some error in the output. At an initial glance, it appears that the function could be a straight line, with one point that apparently has a very large error. This type of point, a lone point far away from the rest of the points in the data set, is called an outlier. The question is: Is the outlier significant? That is, is the outlier merely a gross error in the data set or does it represent some unknown feature of the input space?

This question cannot be answered without taking more data from the same population. Take a look at Figures 2.2 and 2.3. Figure 2.2 represents one possible result of drawing more data from the same population as Figure 2.1. Clearly if this is what the population looks like then the outlier in Figure 2.1 is significant. However, Figure 2.3 represents another possible result - and in this case it is clear that the outlier in Figure 2.1 is in fact the result of some sort of error. Without being able to draw more data points from the population, we can't determine the significance of the outlier.

In most cases, we are unable to draw more data from the population, which leaves us with the difficult question of what to do with the outliers as they occur.

2.1.5 Error Metrics

The issue of deciding what sort of error metric to use is also an important one.

There are many approaches to handling error, and the type of classifier (regression or classification) plays an important role in deciding what type of error metric to use. The other factor that determines how error should be calculated comes from the data itself.

Regression tools tend to use Root of Mean Squared Error (RMSE) as an error metric. Other common options include absolute error and negative log density loss, although many others exist as well. RMSE is used because it tends to accentuate large errors while not ignoring all of the small. There is an entire family of error metrics using p-norms of the form

$$\sqrt[p]{\frac{\sum_{i=1}^{N} \mid x_{observed} - x_{actual} \mid^P}{N}}$$

where N is the number of observations or attempts to classify and P is usually an integer. With P=1, this reduces to computing the absolute error. With P=2 it calculates the RMSE. The higher the value of P, the more emphasis that is placed on the larger errors. As P approaches ∞ the result converges to the largest absolute error.

With classification, you generally have two types of error metrics. The first and most common is simple symmetric error, where the cost of misclassifying is the same as the gain from classifying correctly. However, some problems require non-symmetric error where the cost of misclassifying is more than the cost of classifying correctly. A good example is a medical system that diagnoses a disease such as cancer. Here, classifying someone who doesn't have the disease is pretty bad, but telling someone they don't have it when they in fact do is much worse. Such a system would use an asymmetric classification error.

2.1.6 Bayes Decision Theory

Knowing how much error there is in a given system is only one part of the problem. In the real world, measurements can hardly be considered perfect. There is always some error in a given system, and sometimes the data collection methods used are inexact at best. It would be good if we could determine the lowest possible error that a certain system can give us.

Suppose we have a conveyor belt that carries fruit from one portion of a plant to be separated and sent to two other parts of the plant. We are interested in providing an automatic means of determining which type of fruit it is, say apples or oranges, and having a mechanical arm divide the fruit into two bins. We let ω denote the *state of nature*, with ω_1 meaning that an apple is the next fruit and ω_2 meaning that an orange is the next fruit.

In the beginning, we don't have much information. If we knew that the plant processes as many apples as it does oranges, we would say that the next piece of fruit on the conveyor belt is just as likely to be an apple as an orange. In statistical parlance, this is equivalent to knowing that the *a priori probability* $P(\omega_1)$ of the next piece of fruit being an apple is the same as the a priori probability $P(\omega_2)$ of the next piece of fruit being an orange. These a priori probabilities reflect our knowledge of how likely the oranges and apples are to appear.

Suppose that the only information that we have is these a priori probabilities. Then our decision in every case has to be to pick that piece of fruit with the higher a priori probability. This gives us the decision rule

$$\textit{Decide } \omega_1 \textit{ if } P(\omega_1) > P(\omega_2), \textit{ otherwise decide } \omega_2$$

which decides apple if there are more apples than there are oranges and decides oranges otherwise. In effect we are minimizing the likelihood that we are wrong, and in the long run we will minimize our error.

In most circumstances, however, we can gather more information. For instance, perhaps we can run the conveyor belt over a scale to determine the weight of the fruit. Lets assume we have small apples (say, Macintosh) and large oranges (large Navel oranges perhaps). Further, lets say we know how the weight of each type of fruit is distributed, that is, the *state-conditional probability* $p(x|\omega_1)$ that the fruit has weight

x given that it is in state ω_1 (ie, the fruit is an apple). Then the weight combined with the a priori probabilities should allow us to make more informed decisions. The new decision rule is provided by the *Bayes Rule*:

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)} \tag{2.1}$$

where

$$p(x) = \sum_{j=1}^{2} p(x|\omega_j)P(\omega_j)$$

That is, the correct decision is made by looking at the weight of the fruit, and deciding to choose apple if the product of the a priori probability of the fruit being an apple and the conditional probability of the apple having the given weight is larger than the product of the a priori probability of the orange being the next fruit and the conditional probability of an orange having that weight. The Bayes Rule then gives us the probability that the true state of nature is either an apple or an orange given the current weight of the fruit. Clearly, choosing the higher probability once again minimizes our risk.

Bayes proved that this decision process is optimal. There is no way to classify anything with lower error as long as you know the a priori probabilities and the conditional class densities. He went on to show that this method can handle discrete or random variables with multiple classes and multiple states of nature.

Why, then, is classification even an interesting procedure? If we know the optimal procedure, why look at anything else? The answer is simple - in the real world, we have imperfect information. We rarely know the exact a priori distribution of the state of the world. Nor do we commonly know the conditional class densities. Generally all we have is a set of observations, from which we want to make predictions. Pattern Recognition helps to fill in the gaps and lets us make some effort to create an accurate prediction tool.

Bayes' result is still significant, even when we don't have perfect information, because it provides us with a lower bound on the possible error of a classification problem. We now have a goal - come as close as we can to the Bayes Error Rate, since that is the theoretical minimum that we can achieve.

2.1.7 Decision Boundaries and Discriminant Functions

There are many ways to view or represent pattern classifiers. One way is to use a set of *discriminant functions.* A discriminant function is any function $g_i(x)$ that takes the inputs of the pattern classifier and produces an output value. There is one discriminant function per class and the classifier is said to assign a certain input to the class ω_i if

$$g_i(x) > g_j(x) \text{ for all } i \neq j$$

Discriminant functions are not unique. In fact, they have the property that we can replace any $g_i(x)$ with $f(g_i(x))$ - as long as f is a monotonically increasing function and we apply f to all of the discriminant functions for a particular classifier, the resulting classification remains the same. In particular, for Bayes Classification, we can choose any of the following functions:

$$g_i(x) = P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{\sum_{j=1}^{c} p(x|\omega_j)P(\omega_j)} \tag{2.2}$$

$$g_i(x) = p(x|\omega_i)P(\omega_i) \tag{2.3}$$

$$g_i(x) = log(p(x|\omega_i)) + log(P(\omega_i)) \tag{2.4}$$

Depending on the application, this gives us several choices of functions to use to do our classification. Regardless, the effect of these different decision rules is the same. It divides the input space into decision regions, 1 for each class. These regions are divided by decision boundaries, which divide up the input space into the different classes. If two regions are contiguous, then the equation of their common decision boundary is

$$g_i(x) = g_j(x) \tag{2.5}$$

Decision boundaries are a second, equivalent method of representing pattern classifiers. If we know where the boundaries are that divide the space into regions, we can analyze exactly how different the two classifiers are. We take the decision boundaries and determine mathematically the function differences and integrate to determine the size of the area that is different.

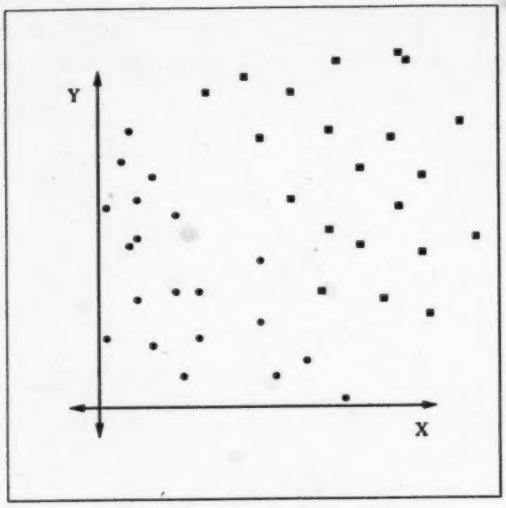


Figure 2.4: Data set

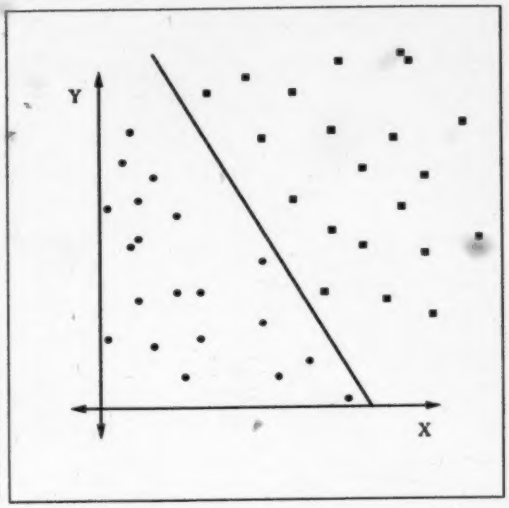


Figure 2.5: Linear Divider

2.2 Methods

Pattern Recognition is a large field. Presented here is an overview of a number of related pattern recognition methods. We start with the simplest of classification methods - Linear Discrimination - and go on to discuss more complicated methods later in this section.

2.2.1 Linear Discrimination

Linear Discrimination is a very simple pattern classification technique that works as follows.

Let say that Figure 2.4 represents a data set that we want to classify. The dots represent one class and the squares another. The X and Y coordinates are the two characteristics that we are using to do our classification. We can create a simple classifier by drawing a line that divides the output into two halves, as in Figure 2.5. Any new input that we receive can be checked against the line. Anything less then the line is a circle, and anything greater is a square.

This method has some obvious advantages. First, it is extremely simple to classify new data points with this method. Second, the method is easy to visualize and explain to people who aren't experts in pattern classification. Third, the method adapts itself

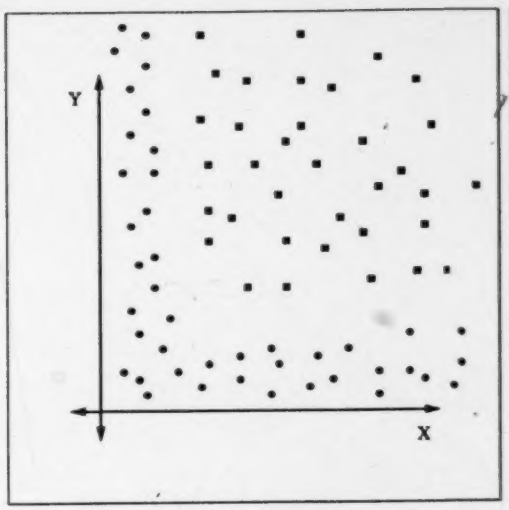


Figure 2.6: No Good Fit

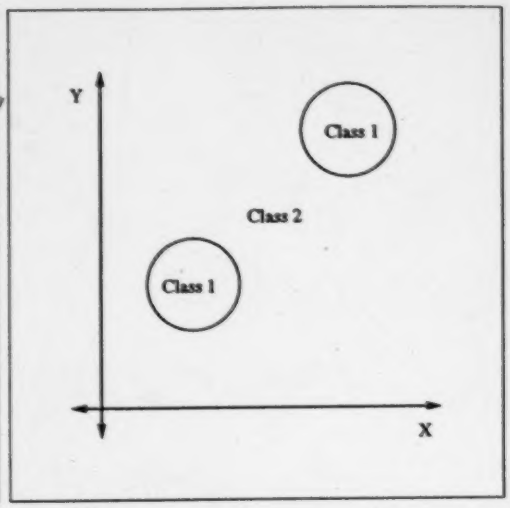


Figure 2.7: No Good Divider

easily to higher dimensions - as long as the two classes are divisible by a hyperplane, this method can be used regardless of the number of dimensions in the input space.

The main disadvantage of this method is also fairly obvious. Simply put, many (if not most) problems are not linearly separable. Figure 2.6 is an example of one such data set, other examples are just as easy to find. A simple trick of mapping quadratic features to a linear space can handle this particular data set, but many data sets are simply not mappable by this method. This limits the usefulness of linear discrimination and makes us look for other methods of classifying our data.

Linear Discrimination presents the simplest possible decision boundary that can differentiate classes. There are methods for creating more complex decision boundaries such as quadratic or cubic functions but generally the methods share the same advantages and disadvantages of linear discrimination - the models are simple and relatively easy to visualize, but many classes aren't differentiable by simple functions. The more flexible the simple decision boundary, the more data sets that can be classified, but the higher the computational costs involved. And there is still the problem that no matter how complex the decision boundary, there are always going to be data sets that cannot be defined adequately such as Figure 2.7, which has two areas of class one contained inside a larger area of class two. No one boundary is going to divide that data set. Clearly, more complex methods are going to be needed.

Figure 2.8: Data set

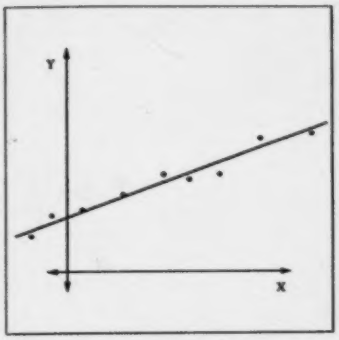


Figure 2.9: Regression

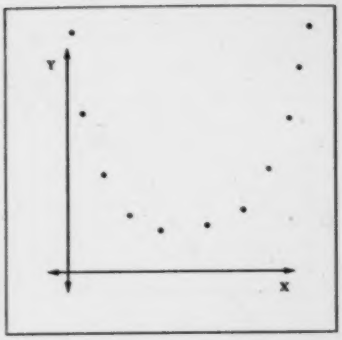


Figure 2.10: No Good Fit

2.2.2 Linear Regression

Linear Regression is the classical regression tool used in just about every scientific field in one form or another. It has its origins in statistics, and forms the basic regression tool that other methods try to improve upon. Linear Regression works as follows.

Take a data set like that of Figure 2.8, where each point represents a known piece of data with one input (the X-axis) and one output (Y-axis). The Linear Regression technique is to draw a line that minimizes the sum of squares distance from the line to the data points. When inputs with unknown outputs are presented to the system, they are plugged into the line equation to determine their outputs. Figure 2.9 shows such a line using the data of Figure 2.9. Multiple input characteristics are handled by using a multi-dimensional line.

The minimum distance from the line to the set of data points can be found in a number of ways. What we want is a linear equation of the form

$$y(x) = a_0 + \sum_{i=1}^{D} a_i x_i \tag{2.6}$$

where the x_i are the inputs in each of the D dimensions, and the a_i are the unknowns that we want to find, with a_0 as a translation factor that allows us to draw lines that don't pass through the origin. The idea is to find a_i that minimize the total square distance from all of the points in the input to the line.

Thus, what we want to minimize is the merit function

$$MF^2 = \sum_{i=1}^{N}(y_i - a_0 - \sum_{k=1}^{D} a_k x_k)^2 \tag{2.7}$$

where N is the number of points we want to fit. Let **M** be the Nx(D+1) matrix composed of each of the input vectors with an additional 1.

$$\mathbf{M} = \begin{vmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1D} \\ 1 & x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & \ldots & x_{ND} \end{vmatrix}$$

Let **B** be a vector of length N composed of the outputs of each of the input points, and let **A** be a vector whose components are the parameters to be fitted, $a_0, a_1, \ldots, a_D$.

The minimum of the merit function (equation 2.7) is found where its derivative with respect to all D+1 parameters a_i vanishes. We are left with D+1 equations

$$0 = \sum_{i=1}^{N} x_{ik} * (y_i - a_0 - \sum_{j=1}^{D} a_j x_{ij}) k = 0, 1, \ldots, D \tag{2.8}$$

where x_{i0} is treated as a 1. These are called the *normal equations* of the least squares problem. They can be solved for the vector of parameters **A** by the standard methods of LU decomposition and back-substitution, Cholesky decomposition or Gauss-Jordan elimination. They can also be solved in a slightly different manner by singular-value decomposition.

The advantages of linear regression are similar to those of linear discrimination. The model is simple and easy to visualize, and once the regression line is known computing unknown outputs is fast and simple.

It also shares similar disadvantages. Figure 2.10 shows a data set for which no line is going to accurately map inputs to outputs. Once again, we turn to more complex methods to try to expand the types of data we can adequately match.

Linear Regression can also be very sensitive to outliers. Figure 2.1 showed us a data set with one prominent outlier. Assuming that Linear Regression is the appropriate tool to use (ie, the data truly is linear), an outlier can skew the line away from the other points to a significant degree. This method doesn't offer the ability

to ignore outliers - all points are treated as significant - and so outliers can skew the line away from its optimal position.

In a similar method to linear discrimination, several more complex methods of essentially the same idea have been attempted. Quadratic functions, cubic functions, and functions of even higher dimensions have been tried with greater and lesser success. Once again, the function that is being described can be arbitrarily complex - simple lines and cubics cannot possibly cover all possibilities. Any chosen function will have data sets that it cannot learn well.

2.2.3 Neural Networks

The term Neural Networks refers to an exceedingly large group of pattern recognition and regression tools. So large is this group in fact that there seem to be no one set of defining characteristics that defines what a neural network actually is. Here are three definitions of the term Neurocomputing given in a graduate-level course on Neural Networks:

> [Neurocomputing is] the technological discipline concerned with information processing systems that autonomously develop operational capabilities in adaptive response to an information environment. [HN90]

> Neurocomputing is the technological discipline concerned with parallel, distributed, adaptive information processing systems that develop information processing capabilities in response to exposure to an information environment. [HN90]

> A neural network is a parallel distributed information processing structure consisting of processing elements (which can process local memory and can carry out localized information processing operations) interconnected via unidirectional signal channels called connections. Each processing element has a single output connection that ...fans out into branches carrying the same signal. Processing within each element must depend only on its current inputs and values it stores - i.e. it is local [HN90]

Clearly, each of these definitions encompasses a different set of information processing systems, with the third perhaps being the most restrictive. So broad a field could easily require an entire thesis in itself merely to touch on the different methods and explain why some were included and others not, and this work can't even begin to do so. Instead, we will describe the most basic of neural networks - Back Propagation Feed-Forward Neural Networks.

A feed forward network is a directed graph with no loops. In such a graph it is always possible to number the vertices in such a way that connections always flow from a lower-numbered vertex to a higher-numbered one. Usually the vertices are arranged in layers, with each successive layer feeding into the next one. There are a lot of different network topologies that can be chosen, but generally they are of the form shown in Figure 2.11. A layer of inputs feeds into one or more hidden layers and then into the output layer. Each of the connections has an associated weight, and each of the vertices of the hidden and output layers has an activation function associated with them. The input to a hidden or output node is the sum of the product of all of the connections to that node and the weight associated with each input. For each hidden or output layer of the network a bias term is included to allow arbitrary linear functions to be described by the sum of the inputs to that layer. The output of the neural network can therefore be seen as a combination of these weights, biases and activation functions.

Let our network have one hidden layer with H vertices, N inputs and M outputs. Then, the inputs to the j-th hidden layer activation function are

$$input(j) = \sum_{i=0}^{N} w_{ij} x_i$$

where w_{ij} is the weight of the connection from input i to hidden vertex j, x_i is the i-th input, x_0 is the value of the bias term and w_{0j} is the weight of the connection from the bias term to the j-th hidden layer activation function. At each hidden node we apply an activation function g to give us

$$output(j) = g(\sum_{i=0}^{N} w_{ij} x_i)$$

These values are summed up at each node in the output layer (output node) and a second activation function h is then applied to each output to give us the final value

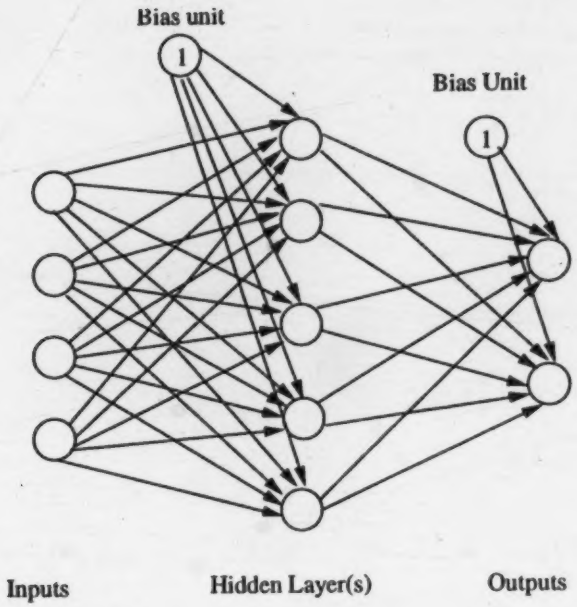


Figure 2.11: Neural Network Topology

of the k-th output as

$$output(k) = h(\sum_{j=0}^{H} g(w_j \times \sum i = 1_N w_{ij} x_i)$$

) g and h are often in practice the same activation function, but this need not be the case.

This model represents only a portion of the various neural network architectures that exist, and yet it has an amazing amount of flexibility. More hidden layers can be added, some connections between layers dropped, and the activation functions can varied. The original activation function was called a perceptron - a function that computes a linear combination of some inputs and returns the sign of the result. It has its basis in biology, where it was believed neurons in the brain functioned quite like the perceptrons in summing up a set of weights and producing a squashed or reduced output. Most biologists no longer believe that the brain is that simple, and the model of neural networks has likewise evolved to encompass a whole range of architectures.

This architecture is not useful as a learning tool if there isn't some mechanism for determining the weights of the various connections. The original manner of "learning"

the weights is called back-propagation [RHW85].

Initially, the weights are set to small, random values. The inputs are then presented to the system and the output(s) are computed. The total error is then computed in a least squares manner. As long as the activation functions are differentiable, we can compute the partial derivative of a given weight with respect to the error, and the weights are updated using a form of steepest descent as

$$w_{ij} \leftarrow w_{ij} - n\frac{\delta E}{\delta w_{ij}}$$

Back-propagation is not the only method of error propagation, nor is least squares the only way of computing error. Variations on the method of error calculation and propagation create even more possible neural network architectures.

In a field as broad as this, we can only talk about the advantages and disadvantages of neural networks in a general way.

The big advantage of neural networks is that, given enough data and enough time, they can approximate any smooth mapping[LF98]. This give them the power to learn a wide range of functions and make sense of a large assortment of data sets.

The big disadvantage of neural networks is their lack of any clear understanding of how the results are obtained. Trying to learn from the results of a neural network is very difficult, and explaining to a lay person what the results of a neural network mean is a difficult task. In addition, it takes a considerable amount of experience to know how many hidden layers and how many nodes in each layer are necessary to learn a given set of data, and often the only way to do so is to try a large number of different architectures until one is found that does the job adequately.

2.2.4 Adaptive Logic Networks

An Adaptive Logic Network is a type of feed-forward multilayer perceptron which uses linear threshold units in a first hidden layer [TAC95], and minimum and maximum nodes in other hidden layers and in the output layer. The computing elements form a tree with the inputs on the bottom and the output on the top. Figure 2.12 shows a typical topology of the network.

An ALN works by building piecewise linear functions and combining them to map the input space. Then, by taking the minima and maxima of the various linear pieces,

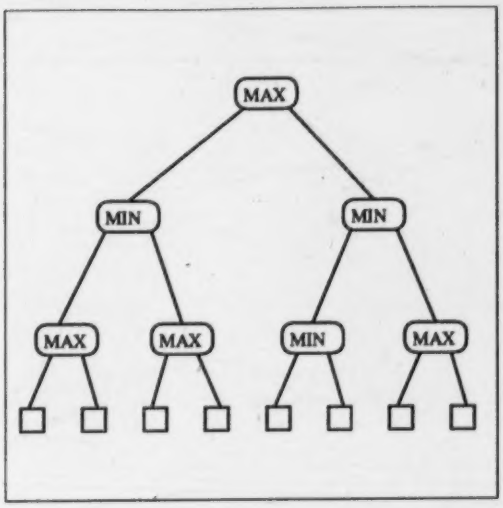


Figure 2.12: ALN Topology

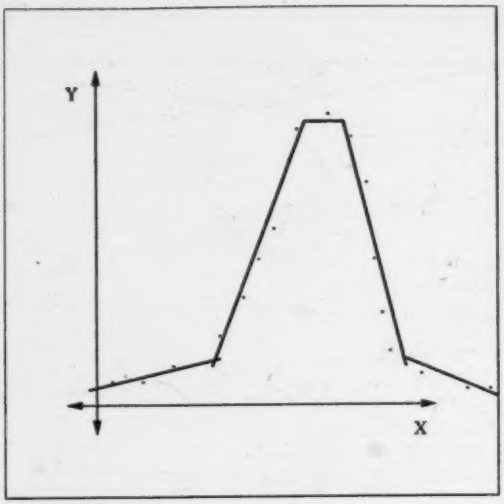


Figure 2.13: ALN Example

a result is obtained. Each linear piece is learned by least squares fitting of the data points in its portion of the data space. When learning, the ALN propagates the error back to the linear piece responsible for that part of the function. This is done by determining which of two functions forms a part of the active surface for the given input vector at maximum and minimum nodes nodes all the way back to the inputs. Figure 2.13 shows an example of what the output of an ALN could look like. ALNs are controlled by the setting of tolerances called epsilons which define how much space each data point can cover. The epsilon is a distance given for each dimension which defines the width of the point along that dimension, and the linear pieces are then maximized with respect to these hyper-rectangles. The points in the ALN are usually jittered as well - that is, for each point in the original space new points are created by holding the output constant and randomly adding a number $\leq$ the epsilon in that dimension to the inputs along each dimension. This creates a new point close to the original with the same output and functions much like an error bar around the data that allows the linear pieces of the ALN to be fitted with more flexibility.

The piecewise linear nature of ALNs makes the tool very flexible. In addition, the system has the ability to incorporate expert knowledge into the system by applying limits to the weights on certain input and by allowing the user to manually control the shape of the decision tree if desired. As well, the system is fairly easy to describe

to people not familiar with statistical methods in general, giving them confidence in its workings.

One of the drawbacks of the system is its concentration on global information rather than local. The tolerances and any constraints placed on the system apply in all areas equally, and do not handle well those input sets with a combination of stable and unstable regions or widely different densities, since all regions are treated the same. This is compensated somewhat by the piecewise linear nature of the system - areas of high instability will tend to have more linear pieces that stable areas - but this does not alleviate the problem completely. As well, the epsilons which are critical to the success of the algorithm must be set by hand. The success of the system therefore ultimately depends on the experience of the user with setting these critical parameters.

2.3 K-Nearest Neighbours

K-Nearest Neighbours is a pattern recognition tool that is primarily used as a pattern classifier, at least in its raw form. The idea of K-nearest neighbours is in widespread use throughout the literature, and in fact the idea is so widespread that in many cases there is no explicit mention of its use.

2.3.1 The idea and the algorithm

The basic idea behind K-Nearest Neighbours is very simple. You find the nearest K points to the current point in the input space, and simply look at their classes, setting the class of the current point equal to the majority class in the neighbours. Typically, Euclidian distance is used to measure closeness, although occasionally the points are normalized first or in some other way massaged to represent some information the user has about the data to be looked at.

In the simple two-class case, this amounts to drawing a boundary between the two classes, but unlike linear discrimination the boundary can be very complex since it is composed of multiple of hyperplanes. KNN is rarely used without enhancements, as the computational costs are very high. In the simplest use of KNN, every single point in the training set is checked for every point in the testing set to see if it is

closer than the current K neighbours. In two or more dimensions, that amounts to a lot of floating point multiplications as distances are calculated.

2.3.2 Early Enhancements

The KNN was first described in an unpublished report by Fix and Hodges [FH51], and the first published results were given by Cover and Hart [CH67] a decade and a half later.

The earliest enhancements to the KNN attempted to make the input set smaller by discarding portions of the input. Hart's Condensed Nearest Neighbour rule (CNN) [Har68] was followed by Gates' Reduced Nearest Neighbour Rule [Gat72] and later improved by Tomek [Tom76]. A related idea called multiedit was developed later by Devijver and Kittler [DK82].

Hart's CNN rule worked very simply. The first step was to divide the data set into two parts which Hart called a grabbag and a store. Step 2 was to classify each point in the grabbag using the store. If the point was classified incorrectly, it was transferred into the store. Step 2 was repeated until all points in the grabbag had been examined and none were transferred to the store, or if the grabbag was empty. The resulting store was then used as the training set and any points in the grabbag were discarded. Gates went slightly further - once he had the store, he classified each point in the store using the store as a training set. Any points in the store that were not needed to classify the edited training set were discarded.

Devijver and Kittler's MultiEdit routine worked in a slightly different manner to get the same end. The set of input points was divided into N subsets, where $N \geq 3$. Each pair of subsets was then chosen with one as the test set and one as the training set. For each such pair of sets the test set was classified using the training set and all misclassified points were deleted. If a complete run through all the pairs deleted no points then the algorithm terminated, otherwise it continued to pair up the sets and run them.

With each of these algorithms a corresponding loss of accuracy occurs as the amount of information in the system decreases with each deleted point. This tradeoff between accuracy and speed was crucial at the time as the computational complexity of the raw KNN algorithm was prohibitively expensive.

2.3.3 Fast KNN

In response to the high computational costs of the KNN, a second set of algorithms was developed that attempted to keep all of the information and eliminate unneccesary computations by using some reasonable rules. Friedman et al [FBS75] developed one such approach, and Fukunaga and Narendra [FN75] developed a branch-and-bound algorithm that was implemented by Kamgar-Parsi and Kanal [KPK85] and modified by Niemann and Goppert [NG88]. The algorithms make the KNN algorithm more attractive by reducing the overall computational costs without impacting the overall classification accuracy.

2.3.4 Weighted KNN

A further enhancement to the KNN algorithm was developed by Dudani [Dud76]. Using a straight-line distance weighting scheme that weighted the nearest neighbour at 1 and the furthest at 0, Dudani computed the unknown class by summing up weights of the neighbours for each represented class. The unknown sample is assigned the class with the highest total weight. Dudani performed a Monte Carlo analysis to show one particular case where his method outperformed the unweighted majority-rules classification. Soon thereafter, Bailey and Jain [BJ78] proved that the asymptotic classification error rate of the unweighted k-nearest neighbour rule is lower than any weighted k-nearest neighbour rule. So, in the limit, the weighted K-nearest neighbour rule will always be less accurate. However, the weighted KNN rule resurfaced as a viable decision rule when Macleod, Luk and Titterington [MLT87] proved that when the number of training samples is finite, the weighted classification rule can outperform the unweighted one.

2.3.5 Modern KNN usage

Today the KNN method is used all over as a method of extracting information from systems. The KNN is used in handwriting recognition [Yan94] and MRI (Magnetic Resonance Imaging) classification [War96]. KNNs have been combined with genetic algorithms [ZH96] and there has been an attempt to combine them with neural networks [HLS95].

As well, the applicability of KNNs has expanded outside of the classification scope and into regression. Attempts have been made by Rasmussen [Ras] and Hastie and Tibshirani [HT96] to make a regression tool using KNN as a base. Rasmussen's approach is a weighted average of the neighbour outputs while Hastie and Tibshirani use discriminant analysis. This has opened up an entire new range of possibilities for the KNN tool, as more and different problems can be tackled with this same idea.

A related area of research, Locally Weighted Leaarning, is reviewed by Atkeson, Moore and Schaal [AMS97]. This area talks about using weighted regression in a local area to do machine learning, using a distance or radius to define the area to be used. This fixed distance is the same regardless of where in the input space you are, as it isn't really KNN, but its incorporates a number of similar ideas in talking about local regions of space and how points in a region can be combined to do regression.

Chapter 3

The KNNR Method

KNNR stands for K-Nearest Neighbours with weighted linear Regression. But what, exactly, does it do? And how does it do it?

3.1 Purpose

The KNNR is a regression tool that takes as input two data files. The first data file, called the training file, provides the set of points used by the K-Nearest Neighbours method. The second file, called the test file or the learning file, provides a set of points for which we wish to find the outputs. The KNNR takes the two files and finds the outputs of the test file using the training file as a set of valid points in the set. We use the K-nearest neighbours idea and, in a similar manner to Rasmussen [Ras] and Hastie and Tibshirani [HT96], apply our own method of using those neighbours - weighted linear regression. We are thus able to look at both regression and classification problems.

3.2 Step 1: Normalization

We begin by reading the data from the training and test files and storing it in memory. The KNNR first normalizes the training set, keeping track of the mean and variance, and then uses these means and variances to normalize the test set. This is done to eliminate any bias caused by changes of scale or transformations applied to the data sets. Because the K-Nearest Neighbours method uses simple Euclidian distance to determine which neighbouring points are closest to the current point, unnormalized

data is very sensitive to the units the inputs were measured in and the magnitudes of the quantities being measured. Normalizing the data emphasizes relative changes in the input range as opposed to absolute changes, eliminating (or at least lessening) the effects of a change in units (from, say, grams to kilograms) or the unbalancing effects of large inputs vs small ones. This has both advantages and disadvantages.

This method is not perfect. By normalizing in every case, we are eliminating one method of human interaction with the system that might produce better results. An intelligent user of an unnormalized system can manipulate the data in such a way as to emphasize the more important inputs and minimize the effect of unimportant data. By normalizing we eliminate this control, but this is a small price to pay for eliminating the two biases by normalization. An intelligent user of the system always has control over what inputs to use and can simply eliminate unimportant ones.

3.2.1 Human Intervention when Evaluating Automated Systems

Perhaps now is a good time to talk about an important issue in automated learning systems, specifically that of human/machine interaction.

Many (if not all) computer learning systems can benefit from the assistance of the humans using them, in a lot of different ways. However, when evaluating the performance of a computer learning tool, one has to be careful not to overstate the success of a computer tool if that success is brought about by human expert knowledge. One has to be careful to evaluate the tool on its own merits, otherwise what you are really evaluating is the success of the human in learning the data, not the computer.

There are no clear lines drawn about what is acceptable and what is not, merely different degrees of acceptability. This author has therefore taken the approach that when comparing the KNNR and ALN with other computer learning systems, all attempts have been made to ensure that no human expert knowledge is exploited by the system. What this means is that every single data file is given to the system in exactly the same way, unmanipulated by human hands. No attempt has been made to tailor the data files to the KNNR by eliminating useless or misleading inputs, adding or deleting outliers manually, replacing variables of one type with variables of another, or in any way attempting to use the author's knowledge of the data to

produce good results.

Clearly, in real life, results are the most important measure of a systems usefulness. This author is not saying that human intervention is inappropriate for a learning tool. In fact, the opposite is true - this author believes that the incorporation of human expert knowledge into computers is almost always going to outstrip any results we can currently obtain through computer use alone. But the point is that when comparing computer learning systems, one has to try to strip away the human interaction to evaluate the computer learning.

So, what does this mean for normalization? Well, it means that we make the decision to normalize in every case because we don't yet have a means of letting the computer determine whether or not to normalize the data. And since it is computer learning that we are investigating, we simply let the computer normalize in every case.

3.3 Step 2: Find the Nearest Neighbours

Once we have normalized the inputs, we begin the KNN part of the KNNR process. For each point in the test set, we look for the K-nearest neighbours, using Euclidian distance as our distance metric. The first question might be "what value do we use for K?" but we defer that discussion to section 3.8. At this point in the algorithm we have a K given to us, and we need to find that many nearest points in the input space.

This can be done exceedingly simply by looking at each point in the data set in turn and merely comparing the distances to each one. This will eventually get the K closest neighbours. However, a small enhancement has been made to make this process more efficient by eliminating some of the points that are too far away.

The basic idea is to divide all of the points up into what we call buckets or bins. Each bucket contains a subset of all of the points in the input space, and each point is in one and only one bucket at a time. Buckets are built by randomly selecting a subset of points (which we call the bucket heads or bucker centres) and then, for each point in the input set we assign it to the bucket that has the closest bucket head. At the same time, we keep track for each bucket the distance to the farthest point

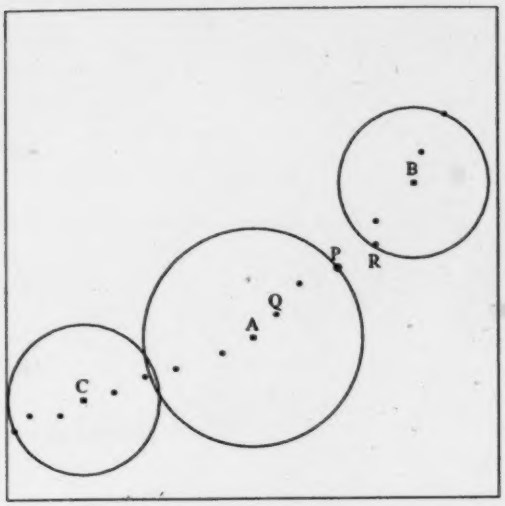


Figure 3.1: Bucket Pruning

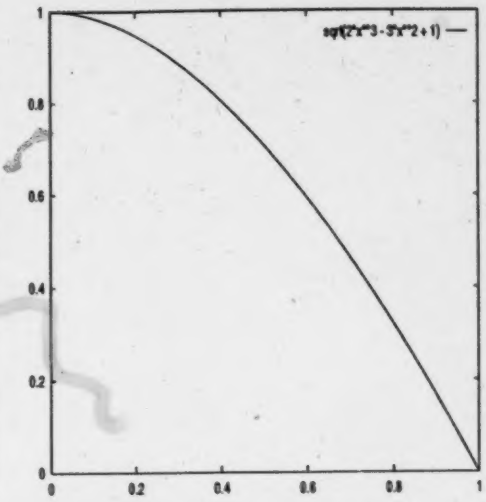


Figure 3.2: Weighting Function

in the bucket, which we call the bucket radius. All of this involves a fair amount of computation. Each of the N points in the data set has to be compared to each of the B bucket heads, so there are $N \times B$ distance calculations to be made. The payoff is that once done the subsequent searches for neighbours can be done quicker. Also, the buckets only have to be built once since points in the input space do not move.

Once the buckets are built, we are ready to find the nearest neighbours. We do this by finding the bucket with the nearest bucket head (the nearest bucket) and picking K+1 points out of it, keeping track of how far the K+1th point is from the current point. Then, we look at the other points in this bucket to see if there are any closer than the current K+1 points, discarding points that are further away, maintaining the nearest K+1 points (it will be clear why we keep K+1 points later, when in section 3.4 we talk about how the points are used to find our new output), and updating the distance to the K+1th point (the neighbour radius) as necessary. Once we have looked at all of the points in a bucket, we find the next-nearest bucket. If the new bucket radius + the neighbour radius is less than the distance to the bucket head, then we discard the bucket, otherwise we go through all of the points in the bucket looking for new nearest neighbours. To help see why this works, see figure 3.1.

P is the current point under consideration. A, B, and C are the three bucket heads for the three neighbouring buckets. In this case, we are looking at K=1, and after

looking at our closest bucket (bucket A) we have the current situation, with Q being the farthest point from us currently. Next we consider bucket B. The distance $\overline{QB}$ is larger than the radius of bucket B + the distance $\overline{PQ}$, which indicates that there might yet be a point in B that is closer to P than Q, so we would look at each point of B in turn. Once we are finished looking at B, R would be our K+1th point. The distance $\overline{QC}$ is greater than the radius of bucket C + the distance $\overline{PR}$, so there is no point in C that is closer than R to Q, and we can discard bucket C and not look at any of its points.

3.4 Using the Nearest Neighbours

So, we have our K+1 neighbours. What then? Traditionally, the K neighbours had classes and each neighbour was looked at to determine the majority class, which was assigned to the new point. What we do is more complex and extends KNN so it can do both classification and regression.

The basic idea is to use the K points as K points in a line by using them to perform a weighted regression. Then we have a hyperplane that gives us the value at the current point. The K+1th point is used by our weighting function to determine the weights on the various points. The whole purpose to weighting the nearest neighbours in the original weighted K-nearest neighbours tool was to emphasize the classes of the nearer points. The same basic idea holds for the weighted regression - we want to emphasize the nearest points in the regression by using a weighting that gives higher weights to the closer points. The K+1th point is called the calibration point, and is used as follows.

Lets say we are determining the weight of a point A, which is one of the neighbours of our new point P, and lets call the calibration point K. Then

$$weight(A) = g(\overline{AP}/\overline{AK})$$

where the function g is our weighting function. The calibration point defines a limiting hypersphere where any other point on that hypersphere or further away has a weight of 0.

3.5 The Weighting Function

The Weighting function that determines the weights of the neighbours can be any function that has a few special properties. The function should be monotonically decreasing in the range [0,1] and it must pass through the point (1,0). We use the function

$$f(x) = \sqrt{2x^3 - 3x^2 + 1}$$

shown in in figure 3.2. The weight functions are restricted to monotonically decreasing functions to give points progressively further away from the new point progressively lower weights. The functions are restricted to those passing through (0,1) to make the output values smooth.

The choice of weighting function will determine what types of inputs will be effective. A weighting function that strongly emphasizes the near points will be more effective for a different set of inputs than one that treats all points more or less the same. The current function was chosen because it satisfied the weighting function criteria, it was easy and simple to calculate, and it de-emphasized distant points while still taking into account a wide radius of points around the current point. Originally, the square root was not included in the evaluation. The square root was added to offset the squared nature of the distance metric. Since the distances are squared quantities of the input vectors, we take the sqrt of the weighting function so that when squared, we get our original function back.

3.6 Smooth Output Values

Obtaining smooth output values is one of the chief reasons to use weighted linear regression in the first place. The idea is that as you move through the space you pick up new points and drop old ones off in a smooth and continuous manner.

Using normal linear regression with KNN would give discontinuities in the outputs. Figure 3.3 represents a one-dimensional data set input in the x-axis and output in the y-axis. Line A is the fit produced by linear least squares using points 1, 2 and 3. Line B is drawn using points 2, 3, and 4. As your input moves away from point 2 towards point 3, there is a place where the input is equidistant to point 4 and point 1.

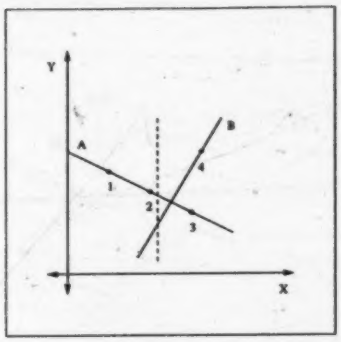


Figure 3.3: Linear KNN

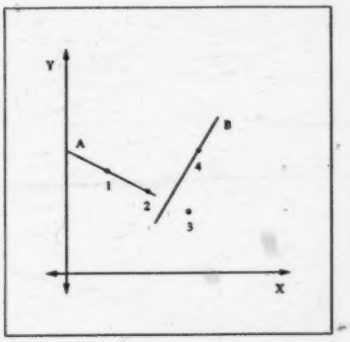


Figure 3.4: Linear Output

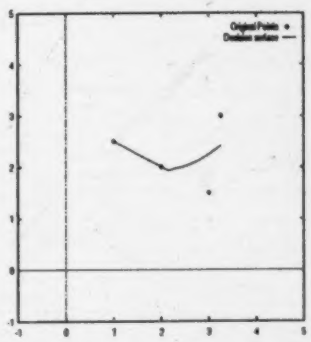


Figure 3.5: KNN Output

This point marks the boundary between line A and line B, and is drawn as a dashed line in the figure. Points to the left of the line use line A and points right of the line use B. The resulting output surface is given in Figure 3.4.

Using weighted linear regression the idea is that as the algorithm moves slowly along the x-axis, the weight of point 1 slowly decreases, reaching 0 when it is the same distance from your x-value as point 4. Then as it moves along the x-axis some more, point 1 has been continuously dropped and point 4 has simultaneously been picked up. Figure 3.5 shows (roughly) the output surface given by the KNN. Note that the surface produced has no discontinuity. The ability to create a smooth output surface means we can use the KNN as a filtering mechanism to eliminate noise from a dataset and gives us some confidence that the method is robust.

There is one further consequence of using weighted linear regression. There is a possibility that two points will be equally distant. In such a case, one of the points will be the calibration point and the other will be the outmost point and will be given a weight of 0. In such a situation, we have to be careful to use enough neighbours that the resulting line is completely determined. Usually, taking dimension+1 neighbours will give us enough points that if two happen to fall on the same surface we still have a completely determined hyperplane.

3.7 Weighted Linear Regression

Weighted linear regression uses exactly the same method as linear regression (see section 2.2.2) except that it performs a pre-processing step to transform the points

into weighted form. This is done very simply by taking each point and multiplying the inputs and output by the weight calculated for that point, then sending the weighted points to the linear regression technique. The KNNR uses svdfit as its linear regression technique.

Svdfit is short for Singular Value Decomposition fit and is one of the many different algorithms that do linear least squares regression. It has two primary characteristics that define its usage - a tolerance and a number of iterations. The tolerance is a measure of how small errors have to be before they are ignored and the number of iterations controls how long the system looks for a solution. Solutions that take longer than iterations attempts are likely to be ill-conditioned. A full explanation is given in [Pre92]

The reason for choosing Singular Value Decomposition is that in the case of an over-determined system, the SVD algorithm gives us the best linear least-squares fit, and if the system is under-determined, svdfit gives us a line that minimizes the error. This allows us to give output even when the system is underdetermined or in the rare case where too many points fall on the limiting hypersphere defined by our calibration point.

Once the weighted regression is complete, we obtain the output by plugging the new point into the hyperplane equation. Once this is done for all of our new points, the process is complete.

3.8 Determining K

There is, however, one last issue that has to be discussed. So far, given a K we know how the KNNR method is applied to determine the new output values. What has not been discussed is how to find K.

The procedure that we used is called the train and test method and is taken from statistical literature. The algorithm begins by splitting the data set into three parts - a training set, a validation set and a testing set. Generally these are not 3 equal parts, although the exact proportions differ from researcher to researcher. The training set is usually the largest, followed by the validation set and then the testing set. This allows for a large number of training examples and a sufficient number of validation

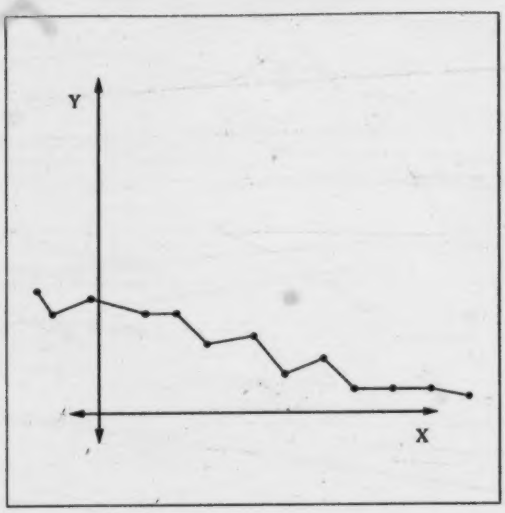


Figure 3.6: Overtraining

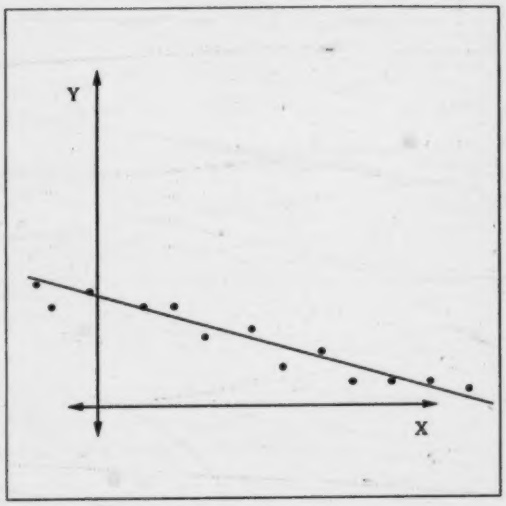


Figure 3.7: Better Generalization

examples to make the test worthwhile.

Once your data has been split up, you put aside the test data. You then train your system using the training data and test its accuracy on the validation data. Once the system is trained to your satisfaction, you try the trained system on the test data, which tests how well your training has generalized.

The reason for this method is simple. Any learning system can be optimally trained to a particular set of data. However, this is no indication of their accuracy on a new data point that comes in. For example, remember how the ALN works by breaking itself into more and more linear pieces until it has satisfied its error requirements. If you let it train long enough and set the error bounds low enough, what you end up with is a connect-the-dots approach that has a single line between consecutive points in the input space. While this learns the given data perfectly, when tested on a new data point the error will generally be large. Figure 3.6 shows this phenomenon, called over-training, while figure 3.7 shows the same data set where a line has been learned that will likely generalize better. Overtraining is particularly bad when there is error in the system, since the system learns the erroneous values correctly instead of finding a middle ground between positive and negative errors. Hence the need for the third data set to act as a test of generalization.

Finding the correct K for the KNNR then becomes a simple procedure. The data

to be learned is broken up into the three files - train, validate, and test. The KNNR uses the train file and the validation file over and over again with different values of K. The K that produces the smallest error in the validation file using the train file is chosen. The combined validation and train files are then used as the new training data and the test set is tried with the optimal number of neighbours to give the algorithm's final results.

Chapter 4

KNNR Tests on Synthetic Data

Every new tool needs to be tested thoroughly to help understand how it works. In this chapter we investigate the properties of the KNNR by experimenting with different data sets that highlight its properties. The purpose of these first tests is to examine the behavior of the KNNR by running tests that isolate a particular feature or variable and test how the KNNR performs with our expectations about its performance.

The synthetic data used was drawn from 3 families of functions.

The first family of functions generated hyperplanes in any number of dimensions. The inputs were real numbers generated randomly in a range of -5000.0 to 5000.0. The output was calculated based on slopes for each of the inputs and a final translation, and then percentage errors were introduced into the inputs of the line. Figure 4.1 shows an example of the function y=2x+4 drawn with 10% error in the overlayed sample points. The error is created by adding a random amount of noise to the inputs of the points in $[-1 \times \%error \times inputvalue, \%error \times input]$.

The second family of functions generated using tanh(x). The functions had random real inputs in the range of 0.0 to 3.0. The output was calculated by $\frac{exp(x)-exp(-x)}{exp(x)+exp(-x)}$ (which calculates tanh(x) at the point x) for each input dimension. No error was introduced. Figure 4.2 shows an example of the y=tanh(x) function drawn and sample points overlayed.

The third family of functions is generated using 1+sin(Ax) functions for various values of A. The functions had random real inputs generated in the range of 0.0 to 10.0. The output was calculated as 1+sin(Ax) for each input dimension. For the error tests error was introduced in the output, otherwise no error was introduced.

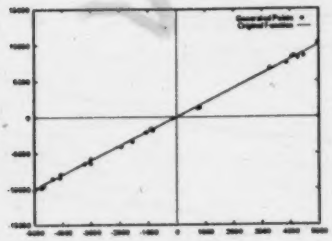


Figure 4.1: 2x+4

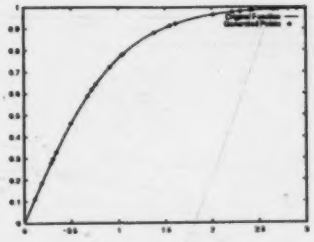


Figure 4.2: tanh(x)

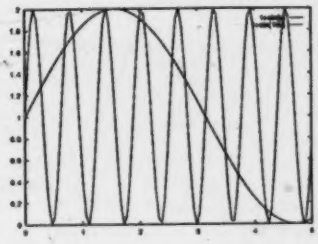


Figure 4.3: 1+sin(Ax)

Figure 4.3 shows examples of two sin curves, y=1+sin(x) and y=1+10sin(x) used often in the synthetic data tests.

These functions were chosen to emphasize particular challenges the KNNR will face. The linear data is the simplest of all possible tests and the addition of error allows us to test how the KNNR performs in uncertain environments. The tanh function gives a good test of how the KNNR handles a constantly changing slope, and the sin functions are good tests of how well the KNNR handles functions with both straight portions and highly curved areas.

4.1 Tests Varying the Numbers of Neighbours

This test is designed to test the effect that using more neighbours has on prediction accuracy. The test consists of generating synthetic data files and running the KNNR with different numbers of neighbours. Synthetic data files consisted of different hyperplanes, sin functions and the tanh function. Each will be looked at in turn.

4.1.1 Hypothesis

We predict that as you initially increase the number of neighbours the accuracy will increase. At a certain point (which will be different for different data sets) it will eventually reach a minimum and the error will increase beyond that point.

4.1.2 Hyperplanes

Four different sets of hyperplanar data were tested. Each test set consisted of data randomly created along the 4 dimensional hyperplane $2x_1 + x_2 - x_3 - 2x_4 + 100$ with

different error percentages. The four tests had error values of 0, 1%, 10% and 50%. 2500 training points were generated for each test, and a test set of 500 points was generated and used for all four tests. The same test set was used for all 4 tests, and it contained no error since what we are looking for is how well the KNNR can learn the underlying function in the presence of error. Values for the number of neighbours went through three ranges. The first range went from 1 to 100 in increments of 1, and the second went from 110 to 240 in increments of 10, and the third went from 250 to 1000 in increments of 50.

4.1.3 Trigonometric Functions

Four different trigonometric functions were used as well. The tanh function was tested in one dimension. Three different one-dimensional sin functions were tested as well - 1+sin(x), 1+sin(5x) and 1+sin(10x). 25000 data points were generated, and five tests were run for each function. The five tests used subsets of 50, 250, 500, 2500 and 25000 training points drawn from the 25000 generated points. The same set of 500 test points was used in all 5 tests.

4.1.4 Results and Conclusion

The following graphs show how the increase in neighbours affects the overall RMSE. The raw RMSE numbers were normalized to effectively show the overall pattern that increased neighbours has independent of the number of sample points used or how much error is present in the linear cases.

The results shown in Figures 4.4-4.7 confirm our hypothesis. Initially, with one neighbour, the error is very high. As the number of neighbours increases, we see a sharp decrease in error at the optimum number of neighbours and then an increase in error as the number of neighbours continues to increase. It is interesting to note just how few neighbours the KNN needs in order to minimize the error. Intuitively, this makes sense - in areas of high curvature a large number of neighbours will quickly drive up error as corners and peaks get cut off.

Figure 4.6 show an interesting behavior for the 50 data point case. The curve drops rapidly and climbs almost as rapidly to a maximum error, then drops down to a minimum. This illustrates the common problem of not having enough data to

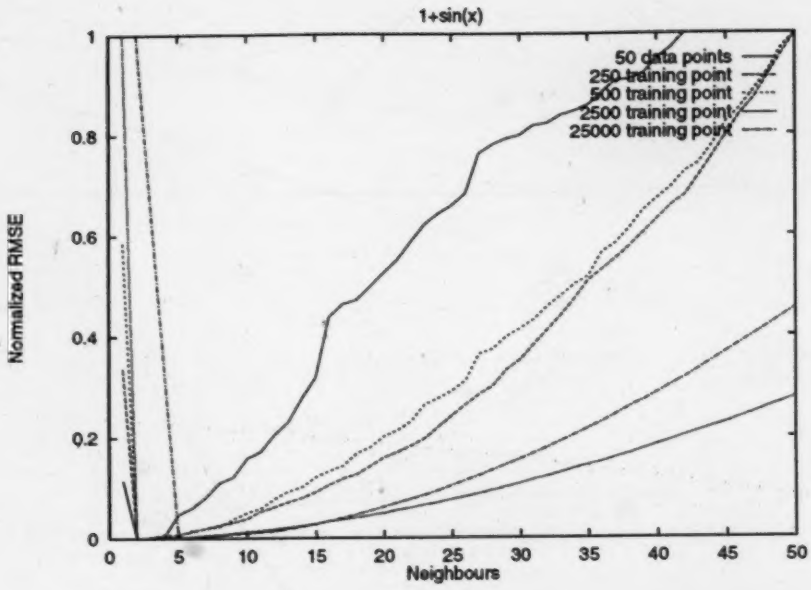


Figure 4.4: Error for 1+sin(x) Neighbour Tests

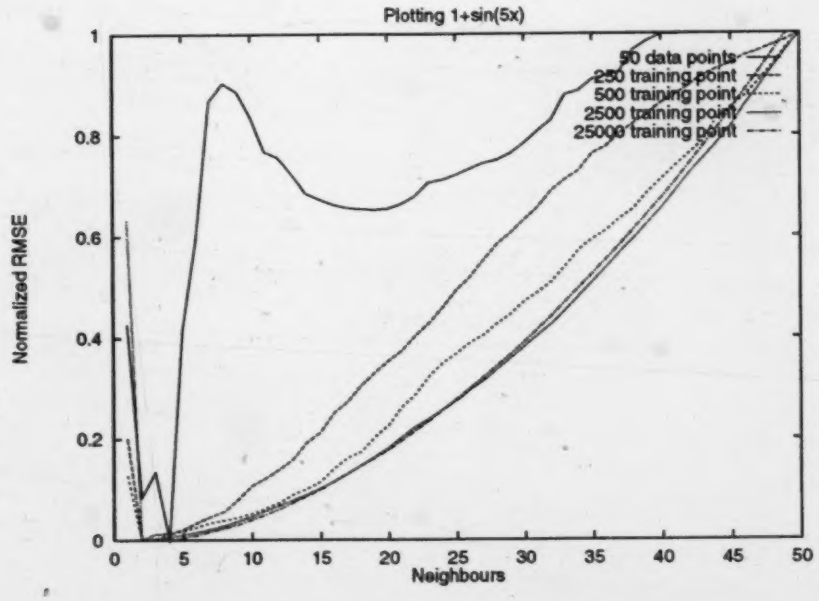


Figure 4.5: Error for 1+sin(5x) Neighbour Tests

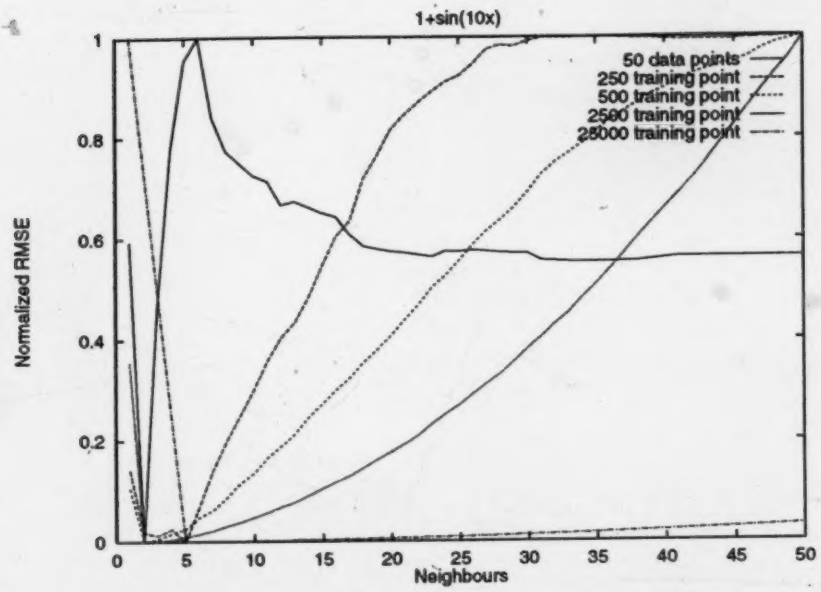


Figure 4.6: Error for 1+sin(10x) Neighbour Tests

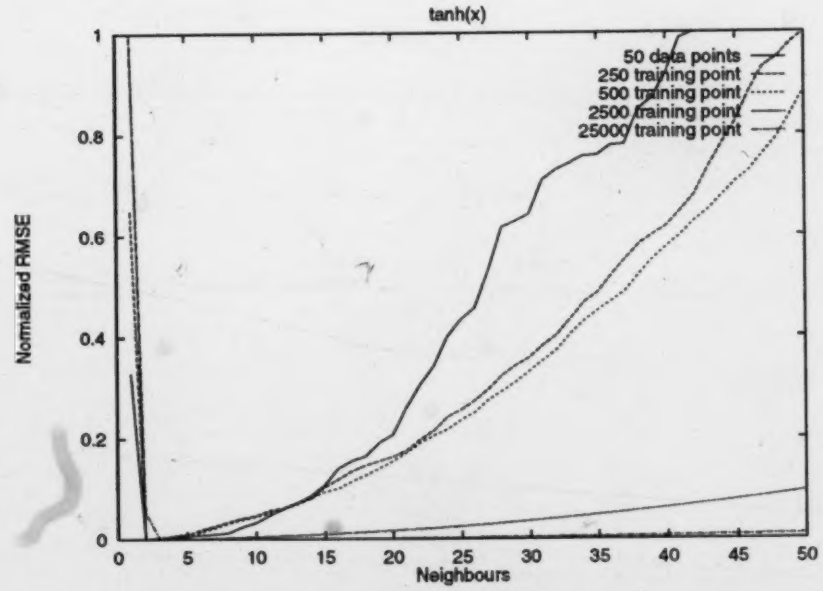


Figure 4.7: Error for tanh(x) Neighbour Tests

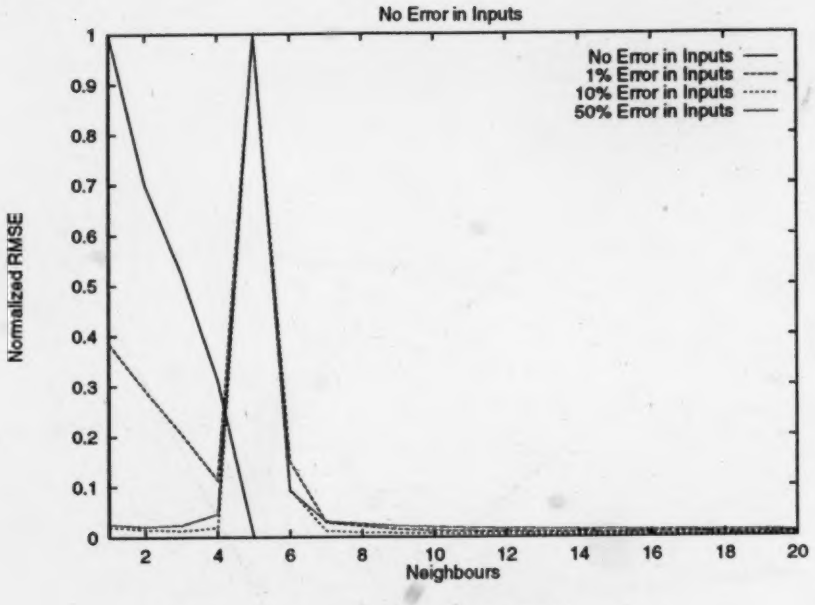


Figure 4.8: Error for Linear Neighbour Tests

describe the state. Figure 4.9 shows what the input data looks like - it is very difficult to see the underlying function (see Figure 4.3) from these few data points. As a result, the KNNR is optimal with either a small number of points, which keeps the points relevant to the local area, or nearly all of the points, which causes the KNNR to lose the KNN part and merely become a weighted least squares regression tool.

Figure 4.5 shows a similar oddity for the 50 data point case, although here there is a bulge in the error that drops and begins to rise again. A similar problem is occuring - the data is not dense enough to describe the space adequately. The drop in error is likely caused by the KNNR picking up points on nearby crests of the sine curve, pulling the line back up and getting closer results to the actual output value.

Figure 4.8 highlights a feature of the KNNR that we did not originally predict, but one that in hindsight we should have expected. Each of these hyperplanes is 4-dimensional. And for 1, 2, 3 and even 4 neighbours we see the expected drop in RMSE. However, in the 5 neighbour case we see a massive jump in RMSE. And beyond this the RMSE drops off almost indefinitely (tests with up to 1000 neighbours were tried, and the RMSE drops very slowly but continues to drop at that point). Why is this the case?

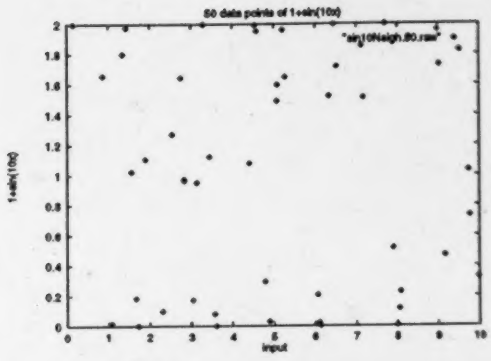


Figure 4.9: Raw 1+sin(10x) Data

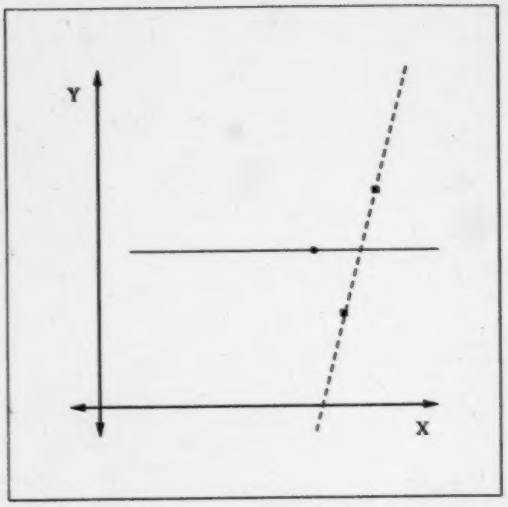


Figure 4.10: Line with Error in Inputs

The answer comes from examining the case where there is no error introduced. Here, we see the expected drop all the way down, and at the 28 neighbour mark the RMSE actually begins to rise slightly as expected. Clearly the strange rise in RMSE at the 5 neighbour mark has something to do with the error introduced in the other 3 examples.

In fact, once the error introduced is carefully examined it is clear why we have a jump in RMSE at the 5 neighbour mark. At the 1, 2, 3, and even 4 neighbour mark the system is not usually completely determined. Because of the distance weighting of the 4 neighbours in most cases the small weight of the 4th (furthest) neighbour causes the system to effectively be underdetermined, and the svdfit algorithm thus gives a "best guess" for the equation of the hyperplane. Once the system becomes completely determined, however, the svdfit algorithm has no leeway and has to give the exact hyperplane determined by the 4 effective points presented to it. Those points have error introduced into them, and the resulting hyperplane can be very far from the actual hyperplane, as in Figure 4.10. This figure shows what can happen when the two nearest points (the squares) have errors in them, one a positive error and one a negative error. The resulting dashed line is far different from the original solid one. Once the number of neighbours starts to increase past the bare minimum, we rapidly have enough points that errors up and down can be compensated for, and

the RMSE drops off very rapidly again.

The fact that the RMSE never appears to climb as the number of neighbours gets very large can also be explained. In this case, the more points you have along the hyperplane, the closer the hyperplane gets to the original and the less impact each individual error has. Each additional point, although it may have error itself, pushes the average error of all of the points to 0 as the limit increases, making the hyperplane truer to the original underlying hyperplane. Since the original surface is hyperplanar, there are no bumps or curves that would get cut off by having too many neighbours, so the numbers of neighbours can increase without apparent limit.

With this explanation, it is unclear why there is a slight rise in error as the number of neighbours increases past 28 in the case where there is no error introduced. Such a problem might be explained away as a rounding problem when the number of points (and hence the number of floating point calculations) gets very large since the magnitude of the increase in RMSE is very small.

4.2 Error Tests

This test is designed to see what the effect of introducing error into the data has on the prediction accuracy.

This test uses similar hyperplanar data to that used in the neighbour tests (Section 4.1). The difference is that a finer mesh of errors has been tested - in this case we tried errors ranging over the same 1% to 50% but looked at 2%, 3%, 4%, etc. Here we hold the numbers of neighbours constant and look at the effect increasing the error has on our prediction error. We also examine the minimum error that the KNNR reaches to see how well it performs. Using linear data means that the results should be the best possible that the KNNR can achieve, since the learning surface is linear.

Once we have some idea what the maximum improvement shown by the system, we then test the KNNR system using sin curves with error introduced, ranging over the same error percentages. We predict that the improvement will be less for curved surfaces, and the more curved the surface the less improvement we will see. In all cases (linear and sine functions) 2500 samples are used in the training set and 500

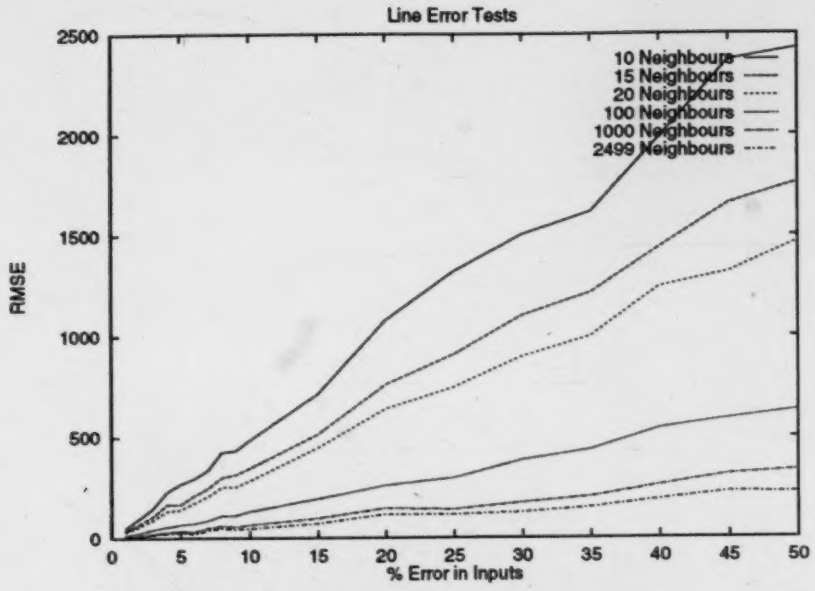


Figure 4.11: Hyperplanar Error Test Results

samples are in the test set.

4.2.1 Hypothesis

We predict that the system will be able to learn the functions to such a degree that the error learned will be no greater than the error introduced, and in fact should be considerably less. We expect that higher numbers of neighbours should have better performance than lower numbers of neighbours, but in all cases the model error should increase as input error increases.

4.2.2 Results and Conclusion

The linear tests gave us the results shown in Figure 4.11. The results were as expected - as the amount of error introduced into the system increases, the RMSE rises. And as we saw in Section 4.1 as the number of neighbours increases, the RMSE drops.

The following table shows the best results obtained by the KNNR. The Error column is given as the maximum percentage error that could be introduced into the system. The linear column gives the best RMSE of the KNNR (which in each case was for 2499 neighbours). The other 3 columns give the best RMSE for the various

sine functions with the given error introduced. The linear columns have a maximum input value of 5000, and the sine columns have a maximum value of 10.

Error(%)	linear (%)	1+sin(x)	1+sin(5x)	1+sin(10x)
1	3.50861	0.00469811	0.0291294	0.812632
2	4.97375	0.0100247	0.0654452	0.195101
3	11.4067	0.0137404	0.116481	0.353858
4	13.9602	0.0193129	0.182907	0.491297
5	15.0481	0.0259115	0.273025	0.578855
6	12.4108	0.0335633	0.355528	0.625001
7	21.5133	0.0459740	0.425094	0.638201
8	33.2391	0.0714939	0.488542	0.640475
9	24.1338	0.0847029	0.581540	0.626845
10	18.8383	0.0980856	0.621076	0.629100
15	39.8058	0.1635120	0.662108	0.642388
20	44.4410	0.222663	0.655313	0.649788
25	94.8598	0.279437	0.664849	0.652528
30	76.3243	0.329299	0.679702	0.657657
35	105.025	0.382082	0.691140	0.667297
40	96.2274	0.429069	0.692964	0.682003
45	102.889	0.468736	0.679270	0.681922
50	118.616	0.493240	0.680413	0.685363

The linear results are very good - the KNNR eliminates at least 90% of the error that was introduced. The 1+sin(x) values are also good - there is a significant reduction in error and the original function is more or less intact. Not surprisingly, as the surface gets more and more bumpy the results are poorer and poorer. This tells us that the system can handle error effectively.

4.3 Training Size Tests

This test is designed to see what the effect of increasing the amount of data available to the system has on KNNR accuracy.

4.3.1 Hypothesis

We predict that as the training size increases, the accuracy for a given number of neighbours should increase, since we have more information about the system. In addition, the minimum achievable error should also decrease when the number of training points increases.

4.3.2 Test Description

This test uses hyperplanar data generated on the hyperplane $2x_1 + x_2 - x_3 - 2x_4 + 100$ with error percentages of 0, 1%, 10% and 50%. A validation set of 500 similar hyperplanes is used for all of the hyperplane tests and contains no error in the same manner as the neighbour tests. Various test set sizes of from 50 to 25000 training points are used and we look at the performance with a variety of numbers of neighbours. For these tests we hold the number of neighbours constant and look at the effect changing the amount of training data has on the system.

4.3.3 Results and Conclusion

The results shown in Figure 4.12 illustrate that our hypothesis is indeed correct - as the training size increases, the accuracy for a given number of neighbours decreases. This graph shows the results for the case where 1% error is present in the hyperplane generated above, other results are analogous. Although the progression is not smooth, as the number of samples increases the accuracy increases until a maximum amount of accuracy (a minimum amount of error) is achieved, at which point the accuracy remains relatively constant. The roughness of the progression is merely a function of the addition of very noisy data points to the samples - a few well-placed highly erroneous data points can create a substantial amount of addition error that is only compensated by the addition of later points with less error. The small rise in error at the beginning of the larger neighbour lines comes about because there are actually less samples than neighbours - in all cases we have an underdetermined system where the "best match" results in better performance than the minimally-constrained systems with error present.

Figure 4.13 shows the minimum RMSE obtained by the system for various sample sizes. Once again we take hyperplanes with 1% error, this time looking to see how the minimum error in the system is affected.

Clearly, as the sample size increases the error decreases. Once again, there are abnormalities in the data since bad data points can be added as easily as good ones when error is present, but the overall trend is clear. The KNNR is capable of handling error.

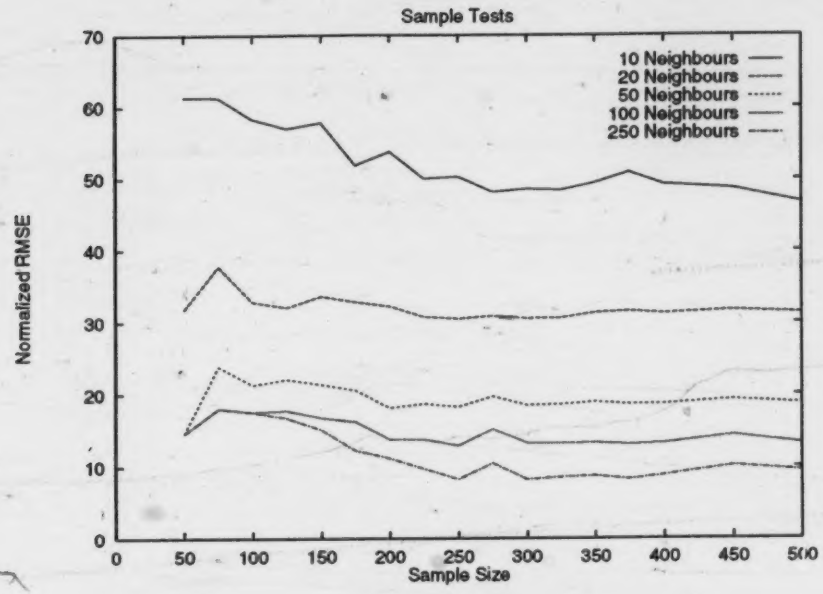


Figure 4.12: Error for Linear Sampling Tests

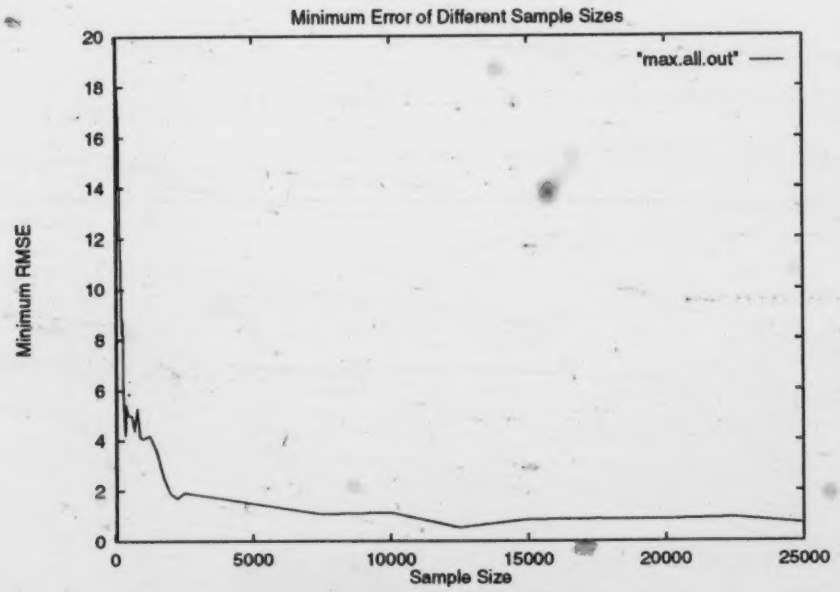


Figure 4.13: Error for Linear Sampling Tests

Chapter 5

Statlog

5.1 Introduction

The Statlog project[Sta] benchmarked over 20 machine learning and classification systems from 1991 to 1994. The systems were compared on their predictive accuracy for each of many different sets of data.

The KNNR's primary use is as a regression tool. However, it can be used effectively for classification as well. The question that we wanted to answer was: How effective a classifier is the KNNR system? While it is impossible to state how effective the KNNR system will be over all data sets, we felt that if we met two criteria we would have a good classifier. First, if we outperformed or performed nearly as well as the standard KNN algorithm, that is a good indicator that the KNNR classifies well. Second, if we perform in the top half of the classification algorithms, that is also evidence that our method works well.

We were able to try the KNNR on five of the Statlog data sets. We now look at the results of each experiment and discuss the performance of the KNNR.

5.2 Methodology

Each of the Statlog data sets came with a standard methodology that was used by all of the learning tools that tried that particular set. The methodologies were mostly the same, and the slight differences will be discussed in the upcoming sections. Those pieces of the methodology that remain constant throughout are described below.

The tests were run on a Sun Ultra-1 workstation. A common tolerance of 1.0e-8

with 250 iterations was used for the svdfit algorithm. In each case, the data sets were taken as is with no attempt at preprocessing. There were two distinct methods of learning used for these 7 data sets - standard train and test and cross validation.

Train and test is the standard method described in Section 3.8. Cross validation is a similar idea, and works as follows. Instead of breaking the data file into 3 distinct files, it is broken up into some number K (K-fold cross validation). K trials of training and testing are then done, where each of the K subfiles is used as the validation set and the other K-1 files are combined to form the training set for that particular trial. The results of the K trials are then averaged to find the optimal set of training characteristics. For each of the datasets that use cross-validation, the number of subfiles to be produced is also given. For train-and-test, the original data files provided by Statlog were already divided into a training set and a test set - the training set was further broken up into a training set and a validation set.

Tests were first done at intervals of 10 neighbours over a range that starts with the dimension of the system and goes upwards of several hundred. Once a broad interval was known to contain the minimum, tests were re-run with much tighter higher and lower bounds at an interval of one neighbour to find the optimal choice of the number of neighbours.

Descriptions of the individual data sets follow.

5.2.1 Australian dataset

The Australian dataset contains data on credit card applications. All of the attribute names and values were changed to meaningless symbols to protect confidentiality of the data. This dataset is interesting because there is a good mix of attributes – continuous, nominal with small numbers of values, and nominal with larger numbers of values. There were originally a few missing values, but those were replaced by the overall median. There are 690 data points given. Each data point consists of 14 attributes (6 Continuous and 8 Categorical) and the output of the system is either a 0 (reject application) or 1 (accept). Where attributes are categorical, the categories are given numerical labels in the order of the relative risk of class 1 (accept). This dataset was learned by 10-fold cross-validation.

5.2.2 Diabetes dataset

The diabetes dataset originally came from the National Institute of Diabetes and Digestive and Kidney Diseases. It contains data on patients who were investigated to determine if they had diabetes according to World Health Organization criteria. The population investigated lives near Phoenix, Arizona, USA. There are 768 data points given and the data set is tested by 12-fold cross-validation. Each data point consists of 8 continuous attributes and the output of the system is either 0 (not diabetic) or 1 (diabetic).

5.2.3 Satimage dataset

The satimage dataset consists of pixels in 3x3 neighbourhoods in a satellite image with each pixel containing information from 4 spectral bands. The aim is to predict the type of land that the pixel image represents. The data set is learned by train-and-test and the training set has 4435 examples and the test set 2000. Each data point consists of the 36 spectral values (real-valued inputs) and one of six output classes. In each data point the four spectral values for the top-left pixel are given first followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on with the pixels read out in sequence left-to-right and top-to-bottom.

5.2.4 Segment dataset

The segment dataset consists of instances that were drawn randomly from a database of seven outdoor images. The images were hand-segmented to create a classification for every pixel. The data is learned by 10-fold cross-validation and the data set has 2310 instances. There are 19 continuous attributes and 7 possible classes of pixel.

5.2.5 Vehicle dataset

The vehicle dataset consists of a set of features extracted from an image of the silhouette of a vehicle. The aim is to predict which of four types of vehicles the image belongs to. The images come from a variety of different angles of the same four vehicles. The data is learned by 9-fold cross-validation and was originally gathered at the Turing Institute in 1986-87 by JP Siebert. The features were extracted from the sil-

houettes by the HIPS (Hierarchical Image Processing System) extension BINATTS, which extracts a combination of scale independent features utilising both classical moments based measures such as scaled variance, skewness and kurtosis about the major/minor axes and heuristic measures such as hollows, circularity, rectangularity and compactness. The four vehicles were a double decker bus, a Chevrolet van, a Saab 9000 and an Opel Manta 400, chosen because of their vastly different profiles. There are 846 data points consisting of 18 continuous attributes and 4 classes of output.

5.3 Results

The following table shows how well the KNNR performed in each of the tests run. The results for the other algorithms are shown for comparison. The KNN and KNNR results are listed on the bottom for easy comparison.

Algorithm	Australian	Diabetes	Satimage	Segment	Vehicle
Cal5	0.131	0.250	0.151	0.062	0.279
Itrule	0.137	0.245	NA	0.455	0.324
LogDisc	0.141	0.223	0.163	0.109	0.192
Discrim	0.141	0.225	0.171	0.116	0.216
Dipol92	0.141	0.224	0.111	0.039	0.151
Radial	0.145	0.243	0.121	0.069	0.307
Cart	0.145	0.255	0.138	0.040	0.235
Castle	0.148	0.258	0.194	0.112	0.505
Bayes	0.151	0.262	0.287	0.265	0.558
IndCart	0.152	0.271	0.138	0.045	0.298
BackProp	0.154	0.248	0.139	0.054	0.207
C4.5	0.155	0.270	0.150	0.040	0.266
Smart	0.158	0.232	0.159	0.052	0.217
BayTree	0.171	0.271	0.147	0.033	0.271
KNN	0.181	0.324	0.094	0.077	0.275
Ac2	0.181	0.276	0.157	0.031	0.296
NewId	0.181	0.289	0.150	0.034	0.298
LVQ	0.197	0.272	0.105	0.046	0.287
Alloc80	0.201	0.301	0.132	0.030	0.173
Cn2	0.204	0.289	0.150	0.043	0.314
QuaDisc	0.207	0.262	0.155	0.157	0.150
Default	0.440	0.250	0.769	0.760	0.750
Cascade	NA	NA	0.163	NA	0.280
Kohonen	NA	0.273	0.179	0.067	0.340
KNNR	0.136	0.238	0.274	0.093	0.256
KNNR rank	2	5	22	17	9
KNN rank	15	22	1	16	11

The performance of the KNNR highlights a couple of strengths and weaknesses. The KNNR performed best when there were only two output classes. The more output classes there were the worse it performed relative to the rest of the field. The KNNR also appears to perform better in cross-validation techniques, although further testing revealed that it performed just as well on the cross-validation techniques as when train-and-test was used, so this may not be significant.

The KNNR clearly holds its own as a classification technique. In three of the five tests it performed in the top 10 of all algorithms tried. In two of the top ten it clearly outperformed the basic KNN technique while it was clearly outperformed by the KNN only once. From this we conclude that we have met our objectives - we outperformed or performed nearly as well as the KNN and we perform in or near

the top half of the classification algorithms - so we can conclude that the KNNR can effectively be used as a classification tool.

Chapter 6

Resampling using KNNR

The purpose of resampling is to populate a data set more thoroughly by taking the data that currently exists and creating a larger set of points that covers more of the input space. We can then use the resampled data to train our system (or another system) to get better results. The KNNR's performance on regression data was good enough to investigate its use as a resampling technique. The basic resampling technique follows.

6.1 The Resampling Method

The input to the system consists of a single data set (the base set), number of neighbours, a radius (the resampling radius), and a constant factor (the resampling factor). The KNNR first normalizes the input set as usual. Then, for each point in the base set, it creates N new points, where N is the resampling factor. Each of these N points is created by adding a random number to each of the components of the original point and then computing the output by the application of the KNNR algorithm for determining the output of a new point. The amount added to each of the inputs falls in the range [-resampling radius, resampling radius]. The effect of this resampling process is to make a larger and more complete data set out of the original while avoiding filling the whole input space. Because of dependencies among the input variables, the data may occupy only a small part of the set of possible inputs.

6.2 Using the ALN

The best benefits of resampling can be found if we can somehow couple the strengths of one learning method with the strengths of another. One of the KNNR's strengths is its ability to adapt to regions of varying local density through the selection of an appropriate number of neighbours. The Adaptive Logic Network described in Section 2.2.4 has as its primary strengths speed and the integration of global knowledge. The way the ALN splits hyperplanes to fit data approaches the regression problem globally, while KNNR attacks the problem locally. We hoped that by using ALNs in combination with the KNNR we would be able to have a faster and more accurate predictor than either method alone.

6.2.1 The ALN Training Method

It is now necessary to have a clear understanding of how the ALN is trained. The basic ALN training method is as follows.

The ALN is used in a series of runs similarly to that of the KNNR (see Chapter 3). A single run of the ALN takes two files - a training file and a validation file. The training file is the file used by the ALN to create its linear pieces. It runs a number of epochs, where in each epoch the linear parts are fitted to the training data and linear pieces are broken into smaller pieces. The resulting collection of linear pieces is then tested on the validation data to find the RMSE. The system runs until the RMSE on the training data is smaller than an acceptable error rate or the maximum number of epochs has been reached. Once either of these termination requirements are met, the system is then tested using the validation file and the resulting RMSE is reported.

To use the ALN, data is divided into three files - a training file, a validation file and a test file. A certain set of ALN parameters (tolerances, min/max slopes, learning rate, number of training runs, and acceptable error rates) are chosen and the ALN is run with these parameters using the validation and training files. The resulting RMSE is then recorded along with the parameters that created it. The parameters are varied by the human user and the ALN is trained anew. Again the parameters are recorded by the human user along with the RMSE. The process is repeated until

an acceptable result is found on the validation set, at which time the ALN is then run using the best parameters on the test set to obtain an unbiased estimate of the error rate.

6.2.2 Our use of the ALN

Keeping focused on computer-only learning, we hold all of the user-adjustable parameters constant except the minimum acceptable error and the input tolerances. The input tolerances are calculated by the KNNR (see below) and the minimum acceptable error is chosen by the human by looking at the KNNR results. This is the only input that the human user has to the system and is done to achieve the best possible generalization of the ALN. It is done by the human because no automatic method of determining the acceptable error has been found.

6.2.3 The Combined ALN/KNNR Method

For the method that combines ALN and KNNR we again restricted our search to computer-only learning, keeping any human intervention to a minimum. The method that we devised for using the KNNR and ALN in concert is as follows.

First, we took the data set and split it into three files. The first two are simple train and validation files, and the third is a test file. The next step is to train the KNNR in the standard way using these training, test and validation files, recording the performance of the ALN and determining the optimal number of neighbours by the result in the validation set. During the validation step, we measure two other quantities - the jitter radius and the ALN radii.

The jitter radius is the maximum radius of the limiting hypersphere formed by the optimal K+1th neighbour (see Section3.4) divided by the dimension-th root of the number of neighbours.

$$JitterRadius = \sqrt[d]{max(R)}$$

where d is the dimension and R is the maximum radius. This distance is a normalized quantity and it represents the maximum amount of space that a single point in the input set covers in one dimension. It is thus the maximum distance that a point

should be jittered during resampling. The ALN radii are the unnormalized jitter radii further divided by the dimension-th root of the jitterfactor,

$$ALNradii = UnnormalizedJitterRadius/\sqrt[4]{jitterfactor}$$

representing the maximum distance that one of the resampled points should cover in any one dimension.

The ALN is then trained as usual (Section 6.2.1) using the combined validation and training sets as input and the test set as the test set, with the epsilon for each input set to the ALN radius. The results are recorded.

The train and validation files are then combined into the base set for the resampling method. Using the optimal number of neighbours found by the KNNR tests and setting the resampling radius to the jitter radius, a jitterfactor is chosen and the KNNR is run in resampling mode with the new base set. This produces a resampled output set. The output set is then used as the input set and the test set is tried by the ALN. Tolerances are set as above and the output RMSE is recorded. We thus get an output RMSE for the three different methods - KNNR, ALN and KNNR+ALN run on the same test set, which makes comparing them simple.

6.3 Results and Conclusions

Figure 6.1 shows the resampling process at work. A random subset of the resampled data is shown overlayed with the original data and the original function. The original data file had 250 data points and the function resampled was 1+sin(5x). The resampled points do not completely cover the entire space, since there are pockets where points are simply too thinly spaced. This may indicate that our resampling radius is a little small, although the results are very good.

A number of different tests were run to see how the resampled data used with the ALN would perform. The tanh function and three sin functions (1+sin(x), 1+sin(5x) and 1+sin(10x)) were tested under various conditions to see how best to use the resampled data.

The results (seen in the table below) were very promising. For the tanh function and 1+sin(x) functions the KNNR outperformed both the KNNR+ALN and the ALN

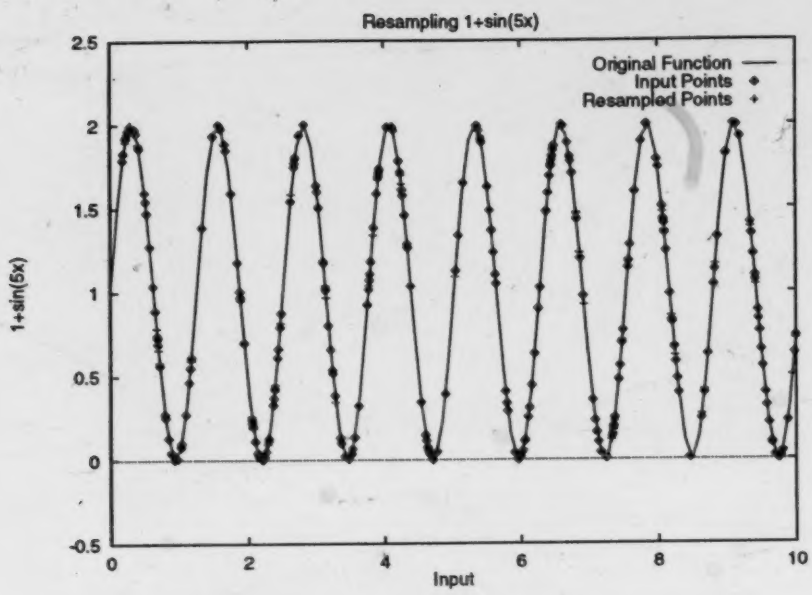


Figure 6.1: Resampling 1+sin(5x)

itself, although in both cases the KNNR+ALN performed better than the ALN. This represents a win for our resampling method. By sacrificing some accuracy we can get a drastic increase in speed, which from our stand on accuracy vs. speed doesn't impress but from a practical standpoint is very useful. There are applications for which the KNNR is simply too slow and while in the future this slowdown is not likely to be a problem, for some practical modern-day problems speed is of the utmost importance.

	KNNR	ALN	Combination
tanh	1.84742e-05	0.000834737	0.000246218
1+sin(x)	0.000383869	0.00342766	0.00350918
1+sin(5x)	0.013745	0.0265242	0.0119124
1+sin(10x)	0.0329672	0.0388779	0.0316295

The results are even better for the 1+sin(5x) and 1+sin(10x) functions. In these cases, the ALN+KNNR performed better than either method, better from both speed and accuracy standpoints. This is an important result, since it shows that we can merge the strengths of the two methods to get two levels of learning - the KNNR and the ALN. Merging their two strengths creates a tool more accurate than either either tool on its own.

Chapter 7

Ship Icing

The final step was to use the KNNR in a real application. The problem chosen involved predicting the rate that ice accumulated on a ship's deck.

Ships travelling in the northern seas of the world meet up with a problem that others in warmer climes do not - ship icing. Ice forming on ships can cause severe imbalance and, in several cases, can lead to the capsizing and sinking of vessels caught in severe storms. Some attempts have been made to build a model to allow scientists to predict the rate at which icing occurs, but these models are still in the development stage. An attempt was made to use both the KNNR and ALNs to try to predict this crucial icing rate.

Ship icing is a difficult process to model for several reasons. The first problem is that ships come in all shapes and sizes. The shape of the ship and its size affect the icing rate and make it difficult to compare results taken from two different ships.

The second and more difficult problem is acquiring accurate data. Getting data on icing rates involves putting a ship and her crew in harm's way, and attempting to make accurate measurements in very adverse conditions. It is difficult to measure parameters such as the sea surface temperature, wind speed and direction, when the winds are up and the sea is tossing water constantly onto the deck of the ship. Measuring the icing rate itself is difficult too, as ice rarely accumulates uniformly and usually contains pockets of trapped brine that add to the mass of the ice but are not themselves frozen. The accuracy of these measurements is therefore poor under the best conditions.

7.1 Previous Work

An attempt at collecting the data needed to build an icing model was undertaken by the Soviet government in the late 1960s. Shipping was a very important contributer to the Soviet economy, and when several ships were lost at sea in a major storm on January 25, 1965, icing was supposed to have been a major factor. This brought about an effort by the Soviet government to collect as much data as they could on ship icing. From 1967 to 1972, the Soviet Ministry of Fisheries conducted a major research program aimed at the collection of field data on the icing and spraying of fishing vessels. The results of this effort were not made available to scientists in the West until 1989, when a paper [ZL89] was released containing a description of the Soviet's efforts to obtain icing data, along with a listing of the 115 data sets recovered. Two further data sets (one by Brown and Roebber (1985) and another by Roebber and Mitten (1987) were provided as well that included 407 data points from a similar study. This larger data set is the one that was used for our testing purposes.

The model built by Lozowski et al. [KKL94] met with some success. A preliminary report given to this author shows how the model was built, and more importantly shows the performance of the model, which was not as good as hoped. The mathematical model tends to over-estimate the values of ship icing. This led to trying the ALN and KNNR to see how well the computer could do.

7.2 Our Work

Both the ALN and KNNR were trained with the data provided by Dr. E. Lozowski. The standard methods of train-and-test were used by both. The ALN was trained by hand, that is, the parameters for the ALN were fine-tuned by a human operator and tested. This could appear to diverge from our stated goal of minimal acceptable human intervention, except that this is a real world problem, and we wanted the best possible results for both tools.

The data used for this experiment is given in an appendix. The inputs to the system are Ship Speed, Wind Speed, Air Temperature (as measured at the surface of the ship), Sea Temperature (the temperature of the air at sea level), the Significant

Wave Height (a measure of the height of the waves in the surrounding sea) and the salinity of the ocean water. The output is the mean icing rate during the icing event.

7.3 Results and Conclusions

Both the KNNR and the ALN performed well on the data set, with the KNNR performing slightly better. KNNR had an RMSE of 0.71629 with 140 neighbours and the ALN had an RMSE of 0.730601. This is a good result for the KNNR showing that even for difficult data sets it can perform as well as and even slightly better than the ALN. It thus confirms that the KNNR is usable as a regression tool on difficult real-world problems.

Chapter 8

Conclusion and Future Work

The KNNR is a regression tool that has shown promise. Its ability to dynamically react to the density of the learned function makes it a powerful tool for learning, and its relative simplicity makes it easy to understand and use.

The KNNR has shown that it is a capable classification and regression tool. Its ability to handle error and its proven performance as a classification tool make it versatile and powerful. And as useful as it is, there are still promising ways of improving the system further and making it even more accurate.

The first avenue of exploration would be to investigate the effect that different weighting functions have on the performance of the KNNR. The choice of what shape the weighting function has will have an impact on the types of data sets that show good performance. A weighting function that weights the nearest points highly but neglects most further points should have different performance than a weighting function that weights most of the points highly, dropping sharply near the limiting hypersphere so most points have a significant impact. Once the effects of the different weighting functions are known, the weighting function could be explored further to see if the KNNR can "learn" which weighting function is best. A simple choice from a preset group of weighting function might be one simple approach, but in the end a system of learning the shape of the best weighting function might gain even better results than those shown here.

A second avenue of further research would be to try the KNNR in conjunction with other types of learning tools. It would be interesting to see if the resampling method could used with Friedman's Multivariate Adaptive Regression Splines [Fri91]

to fill gaps in the input and then fit splines to it, or in combination with a standard Neural Network technique.

A third avenue of exploration would be to explore a method for dynamically changing the number of neighbours during execution. One of the KNNR's strengths is its use of local information - higher density areas use points with a tighter radius than than less-dense areas. It would be interesting to see if this idea could be expanded to include an actual change in the number of neighbours depending on the location in the input space. For instance, a measurement of the planarity of a region could be used to adjust the number of neighbours, allowing for many neighbours in flat areas and fewer neighbours in curved areas.

A fourth avenue of exploration, and one that is perhaps the furthest from realization, would be to get the KNNR to learn which inputs are insignificant, and either eliminate them from consideration entirely or adjust their means and variances to push the points further apart in the less-significant dimension. For instance, doubling the normalized values for a given input would make all of the other inputs relatively more significant when deciding which points were nearest during the Nearest Neighbours phase of the KNNR. A simple version might have the user set a relative weighting scheme that sets the relative widths of the input variables. A more complex system would have the KNNR learn the relative weightings.

Finally, a fifth avenue of exploration related to item four above would be to examine what it means to be the "Nearest Neighbour". Other schemes that adapt the distance metric like those of Lowe [Low95] or Hastie and Tibshirani [HT96] coupled with the weighted regression of the KNNR might show promise.

A final enhancement that could be made to the KNN would be to incorporate the ideas of Niemann and Goppert [NG88] to make the search for K-nearest neighbours more efficient. This would not improve the accuracy of the system, but would make it faster without sacrificing accuracy.

Appendix

This appendix contains the data used in the ShipIcing problem. The seven columns are, in order, Ship Speed, Wind Speed, Air Temperature, Sea Temperature, Significant Wave Height, salinity, and the mean icing rate during the event.

4.6	26.000	-3.9	3.0	8.2	32.0	2.00
3.6	22.000	-13.3	2.5	4.6	32.0	1.20
5.1	20.000	-6.7	3.0	2.7	32.0	0.50
4.6	20.000	-6.7	7.0	0.9	32.0	0.20
4.0	19.033	-3.0	1.0	5.5	32.0	0.04
4.0	18.004	-5.0	2.0	11.0	32.0	0.05
4.0	12.860	-6.0	-1.5	3.0	32.0	0.11
4.0	23.148	-15.0	0.0	7.0	32.0	0.12
4.0	18.004	-12.0	0.5	5.5	32.0	0.13
4.0	12.860	-13.0	0.0	2.0	32.0	0.17
4.0	15.432	-5.0	3.0	4.0	32.0	0.18
4.0	18.004	-4.2	-0.5	5.5	32.0	0.20
4.0	12.860	-3.0	-1.0	4.0	32.0	0.21
4.0	20.576	-5.0	0.0	6.0	32.0	0.22
4.0	10.288	-8.0	0.0	2.0	32.0	0.25
4.0	15.432	-14.0	2.0	4.0	32.0	0.25
4.0	10.288	-10.0	2.0	2.0	32.0	0.28
4.0	10.288	-8.0	2.0	2.0	32.0	0.28
4.0	12.860	-1.0	0.0	3.0	32.0	0.32
4.0	12.860	-7.0	2.0	7.0	32.0	0.32
4.0	21.605	-10.0	2.0	7.0	32.0	0.32

4.0	19.033	-12.0	0.0	5.5	32.0	0.33
4.0	12.860	-2.0	3.0	7.0	32.0	0.35
4.0	28.292	-6.0	0.4	9.0	32.0	0.38
4.0	10.288	-2.0	0.0	4.0	32.0	0.39
4.0	23.148	-3.0	2.0	12.0	32.0	0.40
4.0	12.860	-3.0	0.0	3.0	32.0	0.42
4.0	21.605	-5.0	2.0	7.0	32.0	0.50
4.0	12.860	-2.0	0.0	5.0	32.0	0.51
4.0	15.432	-9.0	2.0	7.0	32.0	0.51
4.0	18.004	-17.0	-1.2	5.5	32.0	0.51
4.0	10.288	-4.0	2.0	2.0	32.0	0.54
4.0	28.292	-10.0	0.0	9.0	32.0	0.54
4.0	12.860	-5.0	0.0	8.0	32.0	0.58
4.0	15.432	-10.0	3.0	4.0	32.0	0.58
4.0	12.860	-4.0	2.0	3.0	32.0	0.64
4.0	18.004	-4.0	-2.0	5.5	32.0	0.64
4.0	12.860	-10.0	1.5	4.0	32.0	0.68
4.0	12.860	-8.0	1.5	6.0	32.0	0.68
4.0	10.288	-10.0	0.0	3.0	32.0	0.69
4.0	12.860	-10.0	0.0	3.0	32.0	0.72
4.0	20.576	-6.0	-1.4	5.5	32.0	0.73
4.0	15.432	-10.0	-1.2	4.0	32.0	0.79
4.0	15.432	-12.0	2.0	4.0	32.0	0.85
4.0	15.432	-3.0	0.0	4.0	32.0	0.85
4.0	12.860	-12.0	2.0	3.0	32.0	0.91
4.0	18.004	-5.0	1.7	5.5	32.0	0.91
4.0	20.576	-12.0	-0.5	7.0	32.0	0.91
4.0	15.432	-12.0	0.0	4.0	32.0	0.97
4.0	19.033	-5.0	2.0	5.5	32.0	1.40
4.0	19.033	-7.0	4.0	5.5	32.0	1.55
4.0	15.432	-18.0	-1.0	4.0	32.0	2.22
4.0	20.576	-8.0	2.5	5.5	32.0	2.29

4.0	20.576	-15.0	2.0	5.5	32.0	10.89
4.0	24.000	-18.0	2.0	3.5	34.0	5.08
4.0	26.000	-9.5	4.0	3.5	34.0	1.39
4.0	15.000	-9.0	3.0	3.5	34.0	0.39
4.1	10.000	-2.8	-1.6	1.1	32.7	0.50
4.9	29.000	-5.5	-1.8	1.1	32.0	2.20
5.0	30.000	-5.5	-1.8	1.1	32.0	2.20
4.1	10.000	-8.9	3.5	2.7	32.0	0.60
4.1	18.000	-12.2	2.5	3.1	32.0	2.70
5.1	20.000	-11.7	2.5	4.6	32.0	3.60
5.1	20.000	-3.3	2.0	4.6	32.0	0.10
5.1	14.000	-2.8	2.5	3.1	32.0	0.10
5.1	20.000	-5.0	3.0	1.8	32.0	1.30
4.6	13.000	-7.8	3.4	4.6	32.0	0.50
4.6	15.000	-8.3	2.7	2.4	32.0	0.20
4.1	26.000	-13.3	3.5	5.5	32.0	0.20
4.1	10.000	-3.3	4.4	1.2	32.0	0.10
3.1	20.000	-3.9	6.5	3.1	32.0	0.20
4.1	13.000	-4.4	1.6	2.4	32.0	0.20
4.1	11.000	-5.0	1.7	2.9	32.0	0.50
3.6	18.000	-2.2	1.9	1.8	32.0	0.50
4.6	20.000	-7.8	6.0	3.2	32.0	0.50
4.6	15.000	-7.2	5.5	3.2	32.0	0.90
5.1	10.000	-5.6	1.0	1.5	32.0	0.20
1.5	16.000	-12.2	4.0	1.5	32.0	6.40
1.5	18.000	-9.4	3.3	1.5	32.0	0.30
5.1	32.000	-6.7	4.0	4.6	32.0	2.00
6.7	20.000	-9.4	4.5	2.4	32.0	1.00
7.7	23.000	-15.0	5.0	3.7	32.0	1.70
4.1	20.000	-12.2	3.5	2.1	32.0	1.90
4.1	28.000	-9.4	3.2	2.4	32.0	1.50
5.1	13.000	-9.4	4.0	1.2	32.0	1.00

4.0	12.860	-3.0	2.0	3.0	32.0	0.03
4.0	15.432	-14.0	0.0	4.0	32.0	0.03
4.0	25.720	-6.0	0.0	18.0	32.0	0.03
4.0	6.173	-1.0	0.0	1.0	32.0	0.04
4.0	19.033	-10.0	0.0	5.5	32.0	0.04
4.0	15.432	-8.0	0.0	8.0	32.0	0.05
4.0	18.004	-12.0	1.0	5.5	32.0	0.05
4.0	18.004	-15.0	2.0	5.5	32.0	0.06
4.0	23.148	-5.0	6.0	6.0	32.0	0.06
4.0	25.720	-10.0	0.0	9.0	32.0	0.06
4.0	25.720	-7.0	2.0	9.0	32.0	0.07
4.0	8.745	-4.0	2.0	2.0	32.0	0.08
4.0	12.860	-6.0	0.0	5.0	32.0	0.08
4.0	15.432	-15.0	0.0	3.0	32.0	0.08
4.0	8.745	-4.0	0.0	2.0	32.0	0.10
4.0	12.860	-9.0	-0.4	3.0	32.0	0.10
4.0	12.860	-6.0	0.0	3.0	32.0	0.10
4.0	23.148	-4.0	-1.0	7.0	32.0	0.10
4.0	10.288	-10.0	0.0	2.0	32.0	0.11
4.0	11.317	-8.0	0.0	3.0	32.0	0.11
4.0	15.432	-5.0	0.5	4.0	32.0	0.11
4.0	18.004	-4.0	-1.6	15.0	32.0	0.11
4.0	25.720	-6.0	0.0	9.0	32.0	0.11
4.0	18.004	-10.0	-1.0	5.5	32.0	0.12
4.0	25.720	-4.0	-1.0	9.0	32.0	0.12
4.0	10.288	-7.0	0.0	2.0	32.0	0.13
4.0	12.860	-6.0	0.0	4.0	32.0	0.13
4.0	15.432	-5.0	0.0	4.0	32.0	0.13
4.0	29.321	-10.0	0.0	11.5	32.0	0.13
4.0	10.288	-6.0	1.0	2.0	32.0	0.14
4.0	10.288	-6.0	0.0	2.0	32.0	0.14
4.0	12.860	-3.0	0.0	3.0	32.0	0.14

4.0	12.860	-5.0	0.0	3.0	32.0	0.15
4.0	15.432	-9.0	0.0	4.0	32.0	0.15
4.0	24.691	-7.0	0.0	9.0	32.0	0.15
4.0	25.720	-4.0	-2.0	9.0	32.0	0.15
4.0	15.432	-15.0	0.0	4.0	32.0	0.16
4.0	20.576	-6.0	0.0	5.5	32.0	0.16
4.0	10.288	-10.0	2.0	3.0	32.0	0.17
4.0	12.860	-8.0	2.0	2.0	32.0	0.17
4.0	12.860	-7.0	0.0	3.0	32.0	0.17
4.0	20.576	-4.0	6.0	11.0	32.0	0.17
4.0	10.288	-7.0	1.0	2.0	32.0	0.18
4.0	12.860	-5.0	1.2	4.0	32.0	0.18
4.0	12.860	-10.0	0.0	8.0	32.0	0.18
4.0	15.432	-5.0	0.0	4.0	32.0	0.18
4.0	20.576	-8.0	2.0	5.5	32.0	0.18
4.0	28.292	-15.0	2.0	9.0	32.0	0.18
4.0	10.288	-8.0	2.0	4.0	32.0	0.19
4.0	12.860	-15.0	1.0	3.0	32.0	0.19
4.0	13.889	-5.0	0.0	6.0	32.0	0.19
4.0	23.148	-12.0	3.0	7.0	32.0	0.20
4.0	25.720	-5.0	-1.0	9.0	32.0	0.20
4.0	10.288	-8.0	0.0	2.0	32.0	0.21
4.0	12.860	-10.0	2.0	5.0	32.0	0.21
4.0	12.860	-9.0	0.0	3.0	32.0	0.21
4.0	12.860	-8.0	-1.0	3.0	32.0	0.21
4.0	15.432	-7.0	0.5	4.0	32.0	0.21
4.0	15.432	-11.0	0.0	9.0	32.0	0.21
4.0	25.720	-4.0	-2.0	14.0	32.0	0.21
4.0	25.720	-2.0	-1.0	12.0	32.0	0.21
4.0	15.432	-10.0	0.0	4.0	32.0	0.22
4.0	18.004	-8.0	0.0	5.5	32.0	0.22
4.0	15.432	-5.0	2.0	7.0	32.0	0.23

4.0	15.432	-2.0	4.0	10.0	32.0	0.23
4.0	11.317	-6.0	3.0	3.0	32.0	0.24
4.0	11.317	-4.0	1.5	3.0	32.0	0.24
4.0	12.860	-14.0	0.0	3.0	32.0	0.24
4.0	15.432	-7.0	0.0	4.0	32.0	0.24
4.0	30.864	-15.0	-1.0	11.5	32.0	0.24
4.0	36.008	-6.0	2.0	14.0	32.0	0.24
4.0	8.745	-7.0	0.0	5.0	32.0	0.25
4.0	12.860	-1.0	-0.4	5.0	32.0	0.25
4.0	15.432	-4.0	0.0	4.0	32.0	0.25
4.0	20.576	-8.0	3.0	7.0	32.0	0.25
4.0	20.576	-5.0	2.0	6.0	32.0	0.26
4.0	21.605	-5.0	-0.5	7.0	32.0	0.26
4.0	23.148	-10.0	0.0	14.0	32.0	0.26
4.0	29.321	-3.0	4.0	11.5	32.0	0.26
4.0	10.288	-3.0	0.0	2.0	32.0	0.28
4.0	18.004	-5.0	2.0	5.5	32.0	0.29
4.0	19.033	-5.0	0.0	8.0	32.0	0.29
4.0	19.033	-6.0	-1.0	5.5	32.0	0.29
4.0	20.576	-12.0	2.0	12.0	32.0	0.29
4.0	12.860	-6.0	0.0	3.0	32.0	0.30
4.0	15.432	-5.0	2.0	6.0	32.0	0.30
4.0	21.605	-3.0	4.0	7.0	32.0	0.30
4.0	23.148	-20.0	0.0	7.0	32.0	0.30
4.0	12.860	-7.0	0.0	3.0	32.0	0.31
4.0	12.860	-12.0	-1.5	6.0	32.0	0.31
4.0	20.576	-3.0	1.0	5.5	32.0	0.31
4.0	7.716	-3.0	0.0	2.0	32.0	0.32
4.0	7.716	-6.0	2.0	2.0	32.0	0.32
4.0	10.288	-8.0	2.0	3.0	32.0	0.32
4.0	10.288	-6.0	2.0	2.0	32.0	0.32
4.0	12.860	-4.0	0.0	3.0	32.0	0.32

4.0	15.432	-10.0	0.0	4.0	32.0	0.32
4.0	18.004	-6.0	2.0	8.0	32.0	0.32
4.0	18.004	-1.0	3.5	5.5	32.0	0.32
4.0	19.033	-5.0	2.0	11.0	32.0	0.32
4.0	20.576	-8.0	4.0	5.5	32.0	0.32
4.0	30.864	-3.0	-1.0	11.5	32.0	0.32
4.0	10.288	-2.0	2.0	2.0	32.0	0.35
4.0	12.860	-5.0	0.0	3.0	32.0	0.35
4.0	15.432	-10.0	2.0	4.0	32.0	0.35
4.0	20.576	-10.0	6.0	11.0	32.0	0.35
4.0	7.716	-11.0	-1.0	3.0	32.0	0.36
4.0	23.148	-4.0	0.0	7.0	32.0	0.36
4.0	10.288	-5.0	0.0	20.0	32.0	0.37
4.0	15.432	-10.0	0.0	7.0	32.0	0.37
4.0	20.576	-7.0	6.0	5.5	32.0	0.37
4.0	18.004	-11.0	-1.0	8.0	32.0	0.38
4.0	20.576	-4.0	-2.0	5.5	32.0	0.38
4.0	30.864	-7.0	4.0	18.0	32.0	0.38
4.0	13.889	-16.0	4.0	3.0	32.0	0.39
4.0	15.432	-9.0	3.0	4.0	32.0	0.39
4.0	15.432	-5.0	0.0	4.0	32.0	0.39
4.0	15.432	-8.0	2.0	6.0	32.0	0.40
4.0	6.173	-3.6	-0.5	3.0	32.0	0.42
4.0	15.432	-7.0	-0.5	4.0	32.0	0.42
4.0	18.004	-4.0	0.0	5.5	32.0	0.42
4.0	18.004	-18.0	-1.0	3.0	32.0	0.42
4.0	23.148	-3.0	0.0	7.0	32.0	0.42
4.0	18.004	-5.0	0.0	5.5	32.0	0.43
4.0	18.004	-6.0	0.0	5.5	32.0	0.44
4.0	10.288	-10.0	0.0	2.0	32.0	0.46
4.0	11.317	-5.0	0.0	3.0	32.0	0.46
4.0	15.432	-10.0	0.9	4.0	32.0	0.46

4.0	16.461	-11.0	0.0	4.0	32.0	0.46
4.0	12.860	-4.0	0.0	2.0	32.0	0.47
4.0	18.004	-6.0	0.0	5.5	32.0	0.47
4.0	20.576	-17.0	0.0	5.5	32.0	0.47
4.0	10.288	-14.0	0.0	5.0	32.0	0.48
4.0	12.860	-4.0	0.0	3.0	32.0	0.48
4.0	12.860	-6.0	0.0	6.0	32.0	0.48
4.0	25.720	-7.0	0.0	9.0	32.0	0.48
4.0	7.716	-7.0	0.0	6.0	32.0	0.49
4.0	10.288	-10.0	0.0	2.0	32.0	0.49
4.0	10.288	-5.0	0.0	5.0	32.0	0.49
4.0	15.432	-10.0	0.0	4.0	32.0	0.49
4.0	15.432	-5.0	0.5	4.0	32.0	0.49
4.0	16.461	-3.0	2.0	4.0	32.0	0.49
4.0	18.004	-4.0	2.0	9.0	32.0	0.49
4.0	19.033	-10.0	-1.0	5.5	32.0	0.49
4.0	15.432	-6.0	0.0	4.0	32.0	0.51
4.0	15.432	-3.0	1.0	8.0	32.0	0.51
4.0	20.576	-3.0	0.0	5.5	32.0	0.51
4.0	20.576	-9.0	0.0	5.5	32.0	0.51
4.0	8.745	-4.0	0.0	2.0	32.0	0.53
4.0	15.432	-6.0	0.0	4.0	32.0	0.53
4.0	15.432	-4.0	0.0	4.0	32.0	0.53
4.0	20.576	-6.0	-1.0	5.5	32.0	0.53
4.0	12.860	-7.0	1.1	4.0	32.0	0.54
4.0	12.860	-7.0	-2.0	9.0	32.0	0.54
4.0	15.432	-6.0	0.0	4.0	32.0	0.54
4.0	18.004	-7.0	0.0	5.5	32.0	0.54
4.0	7.716	-12.0	0.0	4.0	32.0	0.56
4.0	10.288	-8.0	0.0	2.0	32.0	0.56
4.0	15.432	-9.0	0.0	4.0	32.0	0.56
4.0	15.432	-4.0	0.0	4.0	32.0	0.56

4.0	23.148	-4.0	0.0	7.0	32.0	0.56
4.0	18.004	-1.0	3.5	5.5	32.0	0.57
4.0	16.461	-3.0	4.0	4.0	32.0	0.58
4.0	23.148	-10.0	0.0	7.0	32.0	0.58
4.0	23.148	-8.0	0.0	7.0	32.0	0.58
4.0	18.004	-6.0	0.0	8.0	32.0	0.59
4.0	13.889	-4.0	1.0	3.0	32.0	0.61
4.0	10.288	-5.0	0.0	2.0	32.0	0.64
4.0	12.860	-2.0	0.0	3.0	32.0	0.64
4.0	12.860	-8.0	0.0	3.0	32.0	0.64
4.0	12.860	-6.0	0.0	3.0	32.0	0.64
4.0	12.860	-2.0	0.0	3.0	32.0	0.64
4.0	12.860	-9.0	0.0	3.0	32.0	0.64
4.0	14.403	-4.0	2.0	4.0	32.0	0.64
4.0	15.432	-12.0	-0.5	8.0	32.0	0.64
4.0	15.432	-6.0	0.5	4.0	32.0	0.64
4.0	15.432	-6.0	0.3	4.0	32.0	0.64
4.0	15.432	-6.0	0.0	4.0	32.0	0.64
4.0	15.432	-1.0	2.5	4.0	32.0	0.64
4.0	18.004	-18.0	0.0	5.5	32.0	0.64
4.0	18.004	-8.0	0.0	5.5	32.0	0.64
4.0	18.004	-4.0	0.0	11.0	32.0	0.64
4.0	20.576	-9.0	0.0	4.0	32.0	0.64
4.0	20.576	-7.0	-1.0	4.0	32.0	0.64
4.0	25.720	-8.0	0.0	9.0	32.0	0.64
4.0	25.720	-10.0	0.0	14.0	32.0	0.64
4.0	23.148	-15.0	1.0	7.0	32.0	0.69
4.0	15.432	-12.0	0.0	9.0	32.0	0.73
4.0	10.288	-5.0	0.0	2.0	32.0	0.76
4.0	12.860	-18.0	0.0	3.0	32.0	0.76
4.0	15.432	-7.0	-1.0	4.0	32.0	0.76
4.0	20.576	-6.0	-1.0	5.5	32.0	0.76

4.0	7.716	-17.0	0.0	5.0	32.0	0.79
4.0	15.432	-6.0	2.0	4.0	32.0	0.79
4.0	18.004	-14.0	0.0	5.5	32.0	0.80
4.0	10.288	-18.0	2.5	4.0	32.0	0.85
4.0	10.288	-5.0	4.0	4.0	32.0	0.85
4.0	16.975	-12.0	2.0	11.0	32.0	0.85
4.0	18.004	-13.0	-2.0	5.5	32.0	0.85
4.0	18.004	-5.0	2.0	5.0	32.0	0.85
4.0	20.576	-13.0	-1.0	5.5	32.0	0.85
4.0	20.576	-9.0	0.0	14.0	32.0	0.85
4.0	20.576	-7.0	-2.0	5.5	32.0	0.85
4.0	23.148	-10.0	0.0	12.0	32.0	0.85
4.0	25.720	-8.0	-1.0	6.0	32.0	0.85
4.0	25.720	-5.0	0.0	10.0	32.0	0.85
4.0	20.576	-11.0	0.0	18.0	32.0	0.88
4.0	10.288	-12.0	-0.5	2.0	32.0	0.91
4.0	15.432	-6.0	0.0	4.0	32.0	0.91
4.0	18.004	-6.0	0.0	5.5	32.0	0.91
4.0	10.288	-6.0	0.0	2.0	32.0	0.95
4.0	10.288	-7.0	1.5	2.0	32.0	1.02
4.0	23.148	-9.0	0.0	12.0	32.0	1.02
4.0	18.004	-15.0	0.0	5.5	32.0	1.04
4.0	19.033	-9.0	0.0	5.5	32.0	1.06
4.0	9.259	-10.0	0.0	2.0	32.0	1.09
4.0	15.432	-6.0	0.0	4.0	32.0	1.09
4.0	30.864	-9.0	0.0	11.5	32.0	1.09
4.0	12.860	-11.0	0.0	3.0	32.0	1.14
4.0	20.576	-14.0	-0.5	5.5	32.0	1.14
4.0	20.576	-5.0	4.0	5.5	32.0	1.16
4.0	6.173	-21.0	0.0	1.0	32.0	1.27
4.0	10.288	-6.0	0.0	3.0	32.0	1.27
4.0	11.317	-15.0	0.0	3.0	32.0	1.27

4.0	11.317	-9.0	0.0	3.0	32.0	1.27
4.0	15.432	-14.0	-1.0	4.0	32.0	1.27
4.0	15.432	-7.0	1.5	4.0	32.0	1.27
4.0	17.490	-4.0	7.0	11.0	32.0	1.27
4.0	18.004	-3.0	0.0	5.0	32.0	1.27
4.0	18.004	-7.0	0.0	12.0	32.0	1.27
4.0	20.576	-6.0	2.0	5.5	32.0	1.27
4.0	25.720	-12.0	0.0	12.0	32.0	1.27
4.0	25.720	-20.0	0.5	8.0	32.0	1.27
4.0	15.432	-4.0	0.0	4.0	32.0	1.29
4.0	20.576	-18.0	4.0	5.5	32.0	1.37
4.0	12.860	-4.0	0.0	3.0	32.0	1.43
4.0	15.432	-8.0	-1.0	4.0	32.0	1.45
4.0	33.436	-16.0	-2.0	14.0	32.0	1.50
4.0	25.720	-17.0	-0.5	12.0	32.0	1.55
4.0	23.148	-5.0	4.0	7.0	32.0	1.58
4.0	10.288	-8.0	3.0	2.0	32.0	1.59
4.0	15.432	-7.0	0.0	4.0	32.0	1.59
4.0	20.576	-5.0	2.0	5.5	32.0	1.59
4.0	23.148	-11.0	-1.6	7.0	32.0	1.59
4.0	13.889	-10.0	0.0	3.0	32.0	1.65
4.0	16.461	-9.0	2.0	4.0	32.0	1.67
4.0	18.004	-4.0	4.0	5.5	32.0	1.69
4.0	20.576	-11.0	-1.0	5.5	32.0	1.69
4.0	15.432	-12.0	4.0	4.0	32.0	2.12
4.0	12.860	-8.0	1.0	3.0	32.0	2.33
4.0	12.860	-18.0	-0.5	3.0	32.0	2.54
4.0	25.720	-10.0	0.0	9.0	32.0	2.54
4.0	20.576	-0.6	3.0	5.5	32.0	2.83
4.0	33.436	-5.0	0.0	14.0	32.0	2.85
4.0	25.720	-6.0	0.0	8.0	32.0	3.18
4.0	28.292	-14.0	2.0	18.0	32.0	3.18

4.0	23.148	-18.0	2.0	7.0	32.0	6.35
4.0	20.576	-15.0	2.0	5.5	32.0	10.89
4.0	13.000	-7.0	1.5	3.0	34.0	1.02
4.0	21.000	-18.0	4.0	2.0	34.0	2.72
4.0	18.000	-9.5	3.0	3.0	34.0	0.49
4.0	21.000	-6.5	3.0	4.2	34.0	0.54
4.0	30.000	-14.5	2.0	4.0	34.0	4.32
4.0	12.000	-5.0	0.0	2.8	34.0	0.49
4.0	13.000	-18.0	1.0	1.0	34.0	2.85
4.0	15.000	-4.5	5.0	3.0	34.0	0.56
4.0	13.000	-9.5	1.0	4.0	34.0	1.16
4.0	18.000	-9.5	0.0	2.1	34.0	1.27
4.0	10.000	-6.0	0.0	1.0	34.0	0.14
4.0	23.000	-4.0	0.0	3.5	34.0	0.56
4.0	21.000	-8.0	2.5	3.5	34.0	2.29
4.0	18.000	-12.0	1.0	5.0	34.0	1.16
4.0	15.000	-10.0	3.0	4.0	34.0	0.58
4.0	15.000	-13.5	0.0	3.5	34.0	1.52
4.0	15.000	-7.0	0.0	2.0	34.0	0.95
4.0	8.000	-11.0	-1.0	1.5	34.0	0.36
4.0	10.000	-10.0	0.0	3.0	34.0	0.46
4.0	15.000	-8.0	-1.0	3.0	34.0	1.45
4.0	8.000	-16.0	-0.5	1.0	34.0	0.32
4.0	10.000	-12.0	-0.5	2.0	34.0	0.91
4.0	26.000	-5.0	-1.0	5.0	34.0	0.20
4.0	23.000	-9.5	1.0	3.5	34.0	0.42
4.0	10.000	-2.0	1.0	1.5	34.0	0.39
4.0	21.000	-6.0	-1.0	3.5	34.0	0.53
5.4	15.000	-6.0	-1.0	1.0	33.0	1.30
5.4	15.000	-6.0	-1.0	1.0	33.0	1.50
5.4	14.000	-6.0	-1.0	1.1	33.0	1.50
5.0	17.000	-5.8	-1.0	0.9	32.9	1.70

5.0	19.000	-5.5	-1.2	1.0	32.3	1.70
5.0	20.000	-5.5	-1.2	1.0	32.0	1.80
5.0	28.000	-5.6	-1.3	0.9	32.0	1.80
5.0	30.000	-5.5	-1.7	0.9	32.0	1.90
5.0	30.000	-5.5	-1.8	0.9	32.0	2.20
5.0	30.000	-5.5	-1.8	1.0	32.0	2.30
5.0	31.000	-5.5	-1.8	1.1	32.0	2.20
5.0	30.000	-5.5	-1.8	1.2	32.0	2.30
5.0	30.000	-5.5	-1.8	1.0	32.0	2.30
4.9	31.000	-5.5	-1.8	1.2	32.0	2.30
4.9	35.000	-5.5	-1.8	1.0	32.0	2.10
4.9	30.000	-5.5	-1.8	1.3	32.0	2.10
6.6	32.000	-4.4	-1.4	1.0	32.0	0.80
6.4	28.000	-4.1	-1.4	1.0	32.0	1.00
6.4	30.000	-4.1	-1.5	1.0	32.0	1.00
6.4	29.000	-4.0	-1.5	1.0	32.0	0.80
6.6	30.000	-4.0	-1.4	1.0	32.0	0.80
2.0	14.000	-7.5	0.5	4.1	32.0	0.95
2.0	8.000	-4.0	1.0	1.7	32.0	0.15
2.0	14.000	-5.0	1.0	4.1	32.0	0.35
2.0	14.000	-6.0	1.0	4.1	32.0	0.60
2.0	14.000	-7.0	1.0	4.1	32.0	0.70
2.0	14.000	-7.0	0.0	4.1	32.0	0.90
2.0	14.000	-9.0	0.0	4.1	32.0	1.10
0.5	32.000	-20.0	1.0	11.0	32.0	3.25
4.6	12.500	-10.8	0.6	3.4	32.0	0.95
4.1	10.000	-2.8	-1.6	1.1	32.7	0.50
5.4	15.000	-6.0	-1.0	1.1	33.0	1.50
5.0	19.000	-5.6	-1.1	1.0	32.3	1.70
5.0	28.000	-5.6	-1.3	0.9	32.0	1.80
6.5	30.000	-4.2	-1.4	1.0	32.0	0.90
2.0	17.000	-10.0	2.4	4.0	32.0	0.23

2.0	18.000	-10.6	2.4	5.0	32.0	0.44
2.0	10.000	-7.0	2.8	3.5	32.0	0.43

Bibliography

[AMS97] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:1-5:11–73, 1997.

[Bis95] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1st edition, 1995.

[BJ78] R. Bailey and A. K. Jain. A note on distance-weighted k-nearest neighbour rules. *IEEE Transactions on Systems, Man, Cybernetics*, 8:311–313, 1978.

[CH67] T. M. Cover and P. E. Hart. Nearest neighbour classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, Inc., 1st edition, 1973.

[DK82] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* NJ: Prentice Hall, 1982.

[Dud76] S. A. Dudani. The distance-weighted k-nearest-neighbour rule. *IEEE Transactions on Systems, Man, Cybernetics*, 6:325–327, 1976.

[FBS75] J. H. Friedman, F. Baskett, and L. J. Shustek. An algorithm for finding nearest neighbours. *IEEE Transactions on Computing*, 24:1000–1006, 1975.

[FH51] E. Fix and J. L. Hodges. Discriminatory analysis - non parametric discrimination: consistency properties. 1951.

[FN75] K. Fukunaga and P. M. Narenda. A branch and bound algorithm for computing k-nearest neighbours. *IEEE Transactions on Computing*, 24:750–753, 1975.

[Fri91] Jerome Friedman. Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19:1–141, 1991.

[Gat72] G. W. Gates. The reduced nearest neighbour rule. *IEEE Transactions on Information Theory*, 18:431–433, 1972.

[Har68] P. E. Hart. The condensed nearest neighbour rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

[HLS95] Y. S. Huang, K. Liu, and C. .Y. Suen. A new method of optimizing prototypes for nearest neighbour classification using a multi-layer network. *Pattern Recognition Letters*, 16:77–82, 1995.

[HN90] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley Publishing, 1990.

[HT96] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbour classification and regression. *Advances in Neural Information Processing Systems*, 8:290–297, 1996.

[KKL94] L. G. Kachurin, A. M. Kobos, and E. P. Lozowski. A new approach to the theory of icing within a supercooled aerosol flow. unpublished manuscript, 1994.

[KPK85] B. Kamgar-Parsi and L. N. Kanal. An improved branch and bound algorithm for computing k-nearest neighbours. *Pattern Recognition Letters*, 3:7–12, 1985.

[LF98] A. Lapedes and R .Farber. How neural nets work. *Neural Information Processing Systems*, pages 442–456, 1998.

[Low95] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[MLT87] J. Macleod, A. Luk, and D. M. Titterington. A re-examination of the distance-weighted k-nearest neighbour rule. *IEEE Transaction on Systems, Man, Cybernetics*, 17:689–696, 1987.

[NG88] H. Niemann and R. Goppert. An efficient branch and bound nearest neighbour classifier. *Pattern Recognition Letters*, 7:67–72, 1988.

[Pat72] E. A. Patrick. *Fundamentals of Pattern Recognition.* Prentice-Hall, 1st edition, 1972.

[Pre92] William H. Press. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 2nd edition, 1992.

[Ras] C. E. Rasmussen. K nearest neighbours for regression.

[RHW85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propogation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1:318–362, 1985.

[Rip96] B. D. Ripley. *Pattern Recognition and Neural Networks.* University of Cambridge, 1st edition, 1996.

[Sta] http://www.ncc.up.pt/liacc/ML/statlog/index.html.

[TAC95] M. M. Thomas, W. W. Armstrong, and C. Chu. Feasability of using adaptive logic networks to predict compressor unit failure. 1995?

[Tom76] I. Tomek. Two modifications of cnn. *IEEE Transactions on System, Man, Cybernetics*, 6:769–772, 1976.

[War96] S. Warfield. Fast k-nn classification for multichannel image data. *Pattern Recognition Letters*, 17:713–721, 1996.

[Yan94] Hong Yan. Handwritten digit recognition using an optimized neighbor classifier. *Pattern Recogniton Letters*, 15:207–211, 1994.

[ZH96] Q. Zhao and T.. Higuchi. Minimization of nearest neighbour classifiers based on individual evolutionary algorithm. *Pattern Recognition Letters*, 17:125–131, 1996.

[ZL89] W. P. Zakrewski and E. P. Lozowski. Soviet marine icing data. Canadian Climate Centre Report Number 89-2, 1989.